Titre: Title:	Classification de kits d'exploitation avec l'utilisation d'explorateurs de pages
Auteur: Author:	Gabriel Cartier
Date:	2014
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Cartier, G. (2014). Classification de kits d'exploitation avec l'utilisation d'explorateurs de pages [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1394/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/1394/
Directeurs de recherche: Advisors:	Jose Manuel Fernandez
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

CLASSIFICATION DE KITS D'EXPLOITATION AVEC L'UTILISATION D'EXPLORATEURS DE PAGES

GABRIEL CARTIER
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2014

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

CLASSIFICATION DE KITS D'EXPLOITATION AVEC L'UTILISATION D'EXPLORATEURS DE PAGES

présenté par : CARTIER Gabriel
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. DAGENAIS Michel, Ph.D., président
M. FERNANDEZ José M., Ph.D., membre et directeur de recherche
M. PAL Christopher J., Ph.D., membre

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, le professeur José Fernandez, pour son soutien, ses conseils et pour m'avoir initié à la recherche. Je remercie mes collègues du groupe de sécurité des systèmes informatiques qui m'ont aidé de proche ou de loin, soit Joan Calvet, Philippe Blondin et Fanny Lalonde. Je remercie également la Banque Nationale de m'avoir octroyé une bourse qui m'a aidé à la réalisation de ce projet. Finalement, je remercie Jean-François Cartier, Vincent Legault, Fellipe Monteiro, Hubert Guérard et Matthieu Ouellette-Vachon pour leur aide et leurs conseils. Sans eux, ce projet n'aurait pas été ce qu'il est aujourd'hui.

RÉSUMÉ

Malgré l'avancement technologique des systèmes informatiques, la sécurité de ceux-ci demeure un problème récurrent. Plus spécifiquement, dans le cas des virus informatiques, les techniques de protection utilisées permettent seulement des détections *a posteriori* notamment grâce à l'utilisation de signatures des binaires. De plus, avec l'arrivée marquée des kits d'exploitation et du paradigme de logiciel malveillant en tant que service[8], un écosystème de collaborations s'est créé, rendant la tâche de compréhension des pirates encore plus difficile. Nous avons émis l'hypothèse qu'il existe beaucoup d'associations qui permettent de déterminer des groupes comportementaux distincts nous aidant à mieux comprendre l'environnement dans lequel les pirates travaillent. Pour ce faire, nous avons développé un système d'exploration de pages web qui permet de récupérer une multitude d'informations sur le site web, le contenu de la page, le binaire utilisé et bien d'autres informations lors d'une attaque. Nous avons réalisé trois expériences d'exploration qui se sont étendues sur une période de près de 3 semaines où plus de 10,000 explorations ont été réalisées sur près de 3000 URL durant lesquelles nous avons récolté près de 200 GO de données. Notre premier test tentait de déterminer si un mécanisme de géolocalisation permettait aux pirates d'infecter différemment leurs victimes en se basant sur l'emplacement de ceux-ci. L'expérience nous a permis de déterminer que certains mécanismes tel le flux rapide DNS ou la balance de charge modifient le comportement des pages en fonction d'où l'infection a lieu, par contre nous n'avons pu conclure que les infections diffèrent en fonction de la localité de la victime. Le second test voulait établir le temps de vie moyen des pages qui sont utilisées pour l'infection. Grâce à celui-ci, nous avons pu noter deux comportements distincts, soit les virus sont distribués sur des pages légitimes par l'entremise d'une campagne de logiciel malveillant, très souvent de la publicité, sinon les pages sont créées uniquement pour l'infection. Dans le premier cas, les pages restent en vie longtemps, puisqu'une fois l'infection nettoyée, les pages sont légitimes. Sinon, dans le deuxième cas, nous n'avons pu déterminer de temps exact puisque la plupart des pages étaient mortes à leur réception qui est au maximum une heure plus tard. Finalement, notre troisième test cherchait à comprendre si un système de liste noire était utilisé pour limiter les infections. L'expérience a été effectuée en octobre 2013 alors que le visage de la cybercriminalité semblait changer après l'arrestation du développeur du célèbre kit d'exploitation *Blackhole*. Notre expérience nous a permis de constater que notre flux d'URL avait beaucoup changé et ne semblait plus générer aucune infection. Malgré qu'il soit difficile d'attribuer ceci aux évènements, il n'en est pas moins intéressant.

Somme toute, notre système nous a permis de déterminer certains comportements que les

pirates utilisent qui ne nous semblaient pas nécessairement évidents. Par contre, le système nécessite encore beaucoup de travail pour augmenter son efficacité et l'amplitude des données recueillies. Certaines solutions à code source ouvert semblent être une excellente option pour faire évoluer le projet.

ABSTRACT

Despite the technological advancement of computer systems, security remains a recurring problem. More specifically, the computer virus protection techniques used allow only detections *a posteriori*, thanks to the use of binary signatures. Furthermore, the arrival of exploit kits created a new paradigm, namely Malware-as-a-Service[8] where an ecosystem of partnerships is created, making the task of understanding the hackers even more difficult. Our hypothesis was that there are many organizations that work together and that we can identify distinct behavioral groups to help us better understand the environment in which hackers coexist. To do this, we developed a crawling system that allows us to retrieve an important ammount of information on the web pages that deliver viruses.

We conducted three experiments spread over a period of about three weeks where more than 10,000 crawls were conducted on nearly 3000 URLs during which we collected nearly 200 GB of data. Our first test was trying to determine whether hackers use geolocation techniques to infect differently their victims. The experience has allowed us to determine that certain mechanisms such as fast flux DNS or load balancing change the behavior of pages depending on where the infection takes place. We could not conclude that the infections differ based on the location of the victim. The second test would establish the average lifetime of pages that are used for the infection. We noted two distinct behaviors: either viruses are distributed on legitimate pages through a campaign of malicious software, very often through advertising, otherwise the pages are created only for infection. In the first case, the pages remain alive long, since once the infection is cleaned from the website, the pages remain active for their legitimate business. In the second case, we could not determine the exact time as most pages were dead upon receipt which has a delay of at most an hour. Finally, our third test sought to understand if a blacklist system was used to limit infections. The experiment was conducted in October 2013 while the face of cybercrime seemed to change after the arrest of the famous hacker operating the *Blackhole* exploit kit. Our experience has enabled us to see that our URLs' feed had much changed and no longer seemed to generate infections. Although it is difficult to link this to the event, it is no less interesting.

All in all, our system has allowed us to identify certain behaviors that hackers use that were not really obvious. Disadvantages include the system still requires a lot of work to increase efficiency and the amplitude of the data. Some open source solutions seem to be an excellent option to continue the project.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES SIGLES ET ABRÉVIATIONS

API	Application Programming Interface
BGP	Border Gateway Protocol
BHO	Browser Helper Object
CTL	Computation Tree Logic
CTPL	Computation Tree Predicate Logic
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DLL	Dynamic-Link Library
DNS	Domain Name System
GB	Gigabyte
GED	Graph Edit Distance
HTTP	Hypertext Transfer Protocol
IRC	Internet Relay Chat
MaaS	Malware-as-a-Service
MIME	Multipurpose Internet Mail Extensions
NOP	No Operation
PE	Portable Executable
PDF	Portable Document Format
PID	Process Identifier
PCAP	Packet Capture
SaaS	Software-as-a-Service
TCP	Transmission Control Protocol
RPV	Réseau Privé Virtuel
UDP	User Datagram Protocol
URL	Uniform resource locator

LISTE DES TRADUCTIONS

Adware	Publiciel
Autonomous system	Système autonome
Application programming interface	Interface de programmation
Binary	Binaire
Botnet	Réseau de robots
Browser helper object	Objet d'aide au fureteur
Cluster	Grappe
Clustering	Regroupement
Control flow	Flux de contrôle
Data mining	Exploration de données
DNS query	Interrogation DNS
Drive-by-download	Téléchargement forcé
Fast flux	Flux rapide
Framework	Cadre d'application
Honeypot	Pot de miel
Hooking	Accrochage
Hostname	Adresse Internet
Landing site	Site d'infection
Malware	Logiciel malveillant
Malware-as-a-Service	Logiciel malveillant en tant que service
Obfuscation	Obscurcissement
Packer	Protecteur
Parsing	Décomposition analytique
Patch	Correctif
Peer-to-peer	Pair-à-pair
Plugin	Module d'extension
Program slicing	Découpage de programme
Ransomware	Rançongiciel
Rootkit	Outil de dissimulation d'activité
Service pack	Paquet de services
Snapshot	Instantané
Software engine	Engin logiciel
Software-as-a-Service	Logiciel en tant que service

Spam	Polluriel
Training Set	Ensemble d'entraînement
Trojan	Cheval de Troie
Virtual Private Network	Réseau privé virtuel
Widget	Gadget logiciel

CHAPITRE 1

Introduction

Depuis l'existence de l'informatique telle que nous la connaissons aujourd'hui, il y a toujours eu des gens cherchant à trouver des failles pour exploiter les systèmes de façon malicieuse. Le terme logiciel malveillant définit un programme qui exécute une suite d'instructions dans le but d'exploiter un système informatique à l'insu de l'utilisateur infecté[104]. En effet, le terme *virus informatique* fut introduit dans les années 80 alors que Fred Cohen a écrit un programme qui lui permettait d'obtenir les privilèges administrateurs sur les machines infectés[15]. Considérant l'évolution de l'informatique, il est donc logique que les gens développant des virus ou, plus généralement, des logiciels malveillants aient, eux aussi, progressé. Les pirates informatiques utilisent aujourd'hui plusieurs moyens pour infiltrer les systèmes : les virus, les vers informatiques, les outils de dissimulation d'activité, les chevaux de Troie, les réseaux de robots et les logiciels espions pour n'en nommer que quelques-uns.

Avec l'explosion qu'internet connait depuis les années 2000 : la multiplication des usagers, l'apparition des réseaux sociaux ou l'arrivée des téléphones intelligents et tablettes, le développement de logiciels malveillants est maintenant une menace importante du monde informatique. L'industrie du logiciel malveillant est aujourd'hui une industrie qui coûte des centaines de milliards de dollars annuellement[17]. En effet, 556 millions de personnes sont affectées annuellement par le cybercrime générant une perte de 110 milliards de dollars américains[17].

Le cybercrime est maintenant présent dans toutes les sphères de l'informatique, des réseaux sociaux aux courriels, en passant par les téléphones cellulaires[62]. Étant donné l'immense portée des logiciels malveillants, et comme ce paradigme est très récent et donc peu étudié, pour cette étude nous allons plutôt nous concentrer sur les kits d'exploitation. La méthode la plus répandue d'infection des usagers se fait via des téléchargements forcés. Cette technique vise à exploiter différentes composantes de logiciels, communément les modules d'extension de fureteurs, pour télécharger et installer des logiciels malveillants à l'insu de l'usager et, dans certains cas, sans même aucune interaction avec celui-ci. Les kits d'exploitation sont souvent vendus ou loués de la même façon que des logiciels en tant que service (SaaS) [107]. L'utilisation de ce paradigme dans le cybercrime amena la création du terme logiciel malveillant en tant que service (MaaS)[74]. Les criminels offrent donc de payer pour l'utilisation de kits d'exploitation et chargent leurs clients par le nombre d'infections qu'ils

obtiendront via le service[50]. Dans cette optique, le présent projet vise à acquérir de l'information sur les kits d'exploitation et, avec des techniques d'exploration de données, analyser les comportements typiques des pirates dans le but d'aider à mieux comprendre l'écosystème à travers lequel plusieurs groupes collaborent.

Le second chapitre du présent document présente la problématique et l'hypothèse qui guide la recherche ainsi que les objectifs. Le troisième chapitre est une mise en contexte en présentant l'état de l'art en terme d'analyse de logiciels malveillants. Ensuite, le quatrième chapitre discute de différentes techniques d'exploration de données utilisées pour découvrir des grappes comportementales des différentes familles de logiciels malveillants. Les cinquième et sixième chapitres présentent notre méthodologie et exposent ensuite les résultats obtenus. Finalement, le septième et dernier chapitre présente la synthèse des travaux réalisés ainsi que les travaux futurs.

1.1 Problématique

Malgré la grande efficacité de détection des logiciels antivirus présentement offerts sur le marché, la majorité ne fait que des détections *a posteriori*. En effet, la méthode classique de détection de virus utilise la signature numérique des binaires. Cette méthode génère une signature du virus, plus précisément un identifiant du virus, en se basant sur le contenu de celui-ci. Avec les nouvelles méthodes d'obscurcissement à l'aide de protecteurs de fichiers, la technique traditionnelle de détection de signature est donc beaucoup moins efficace. Les pirates sont maintenant capables de générer une infinité de variations de leur virus qui passeront inaperçues aux yeux des antivirus, car ils ne respectent aucune signature connue. Les autorités s'intéressent elles aussi beaucoup à la cybercriminalité, qui représente maintenant une perte importante de revenus dans la plupart des pays du monde. En effet, les coûts du cybercrime aux États-Unis en 2012 pour les entreprises sont estimés à plus de 15 milliards de dollars américains en défense et protection. Par contre, nous pouvons remarquer qu'environ 80% de ces coûts ne servent qu'après la détection de l'attaque. Ceci est corroboré par Anderson et coll. qui démontre que l'argent dépensé en défense est presque négligeable par rapport aux pertes subies par une attaque[1]. Il est donc évident que le gouvernement est, tout comme les compagnies d'antivirus, plutôt réactif quant à sa défense contre le cybercrime. Une des raisons est que les autorités n'ont pas de vision globale du problème. La criminalité sur internet est un problème mondial et il est donc important que les pays unissent leurs forces pour tenter d'y remédier. Le milieu de la recherche est un bon moyen de trouver des solutions à un problème d'une telle ampleur. Malheureusement, les chercheurs ne possèdent

pas d'infrastructure globale leur permettant de suivre, d'analyser ou même de partager toutes les informations qu'il leur est possible de récolter. De plus, avec l'évolution de l'internet, les schémas d'attaques ont eux aussi beaucoup changé. La plupart des attaques sont maintenant déployées rapidement et peuvent être dynamiquement modifiées pour éviter des détections. Ceci est particulièrement vrai dans le cas des kits d'exploitations qui ne cessent de croitre en utilisation chez les pirates informatiques. Lorsqu'une attaque est déployée à partir d'une page web, il y a beaucoup plus que seulement l'exécutable (le virus) à analyser. Aujourd'hui, l'infrastructure d'infection représente un défi aussi grand que le développement de binaire. Pour éviter les détections et maximiser les infections, les pirates informatiques ont recours à des systèmes complets qui nécessitent un énorme travail d'ingénierie à gérer. Avec la venue du paradigme de *MaaS*, nous pouvons croire que les gens qui développent et distribuent des virus travaillent maintenant dans un écosystème où plusieurs groupes travaillent ensemble pour maximiser le rendement de chacun. En effet, tout ce qui se trouve autour du déploiement du virus est tout aussi important et est souvent très peu analysé. Pour n'en nommer que quelques-uns, la position géographique du serveur, l'hébergeur web, le contenu de la page ou les scripts pour la page sont tous des éléments qui nous permettent de mieux comprendre le comportement des pirates.

Dans le cadre de notre étude, nous nous intéressons spécifiquement aux kits d'exploitation et leurs méthodes de déploiement. Les méthodes de déploiement représentent globalement les informations de la ou les pages qui pointent vers le virus, le contenu de ces pages, les informations relatives aux pages ainsi que les informations du binaire. Malgré que les recherches sur les logiciels malveillants soient nombreuses, la plupart tentent d'améliorer les détections ou encore de résoudre le problème de détection *a posteriori* des virus. Bien que plusieurs des méthodes proposées soient efficaces, elles analysent l'exécutable directement et n'incluent pas l'environnement dans lequel l'infection s'effectue. En n'analysant que le virus lui-même, les chercheurs concernés n'ont pas de vision globale sur les données et ne peuvent pas suivre leur évolution sur le web. À l'opposé, quelques recherches se sont intéressées aux pages web qui infectaient des utilisateurs. Un peu comme pour l'analyse du binaire, les recherches sont plutôt orientées vers une analyse des pages et de leur contenu, et le font seulement de façon statique. Bien que quelques recherches intègrent les différentes composantes d'une attaque pour des analyses, elles ne sont pas nombreuses et mettent beaucoup d'emphase sur des analyses statiques c'est-à-dire l'analyse du contenu d'une composante et non pas de son comportement lors d'une exécution. Notre recherche innove dans le sens où nous analysons aussi le comportement dynamique d'une infection, chose qui est peu analysée. Nous tentons donc de comprendre plus spécifiquement le comportement typique d'un déploiement d'attaques

en analysant les méthodes utilisées pour y arriver. Notre analyse dynamique ne se limite pas qu'au binaire, nous tentons aussi de comprendre les différents mécanismes de déploiement des pages ainsi que les différents moyens utilisés pour modifier ces pages. Nous voulons aussi déterminer certaines corrélations entre ces différentes techniques, pour déterminer plus facilement des comportements typiques.

1.2 Objectif

L'objectif général du projet de recherche est de mieux comprendre le comportement des pirates informatiques qui déploient des virus sur internet, pour éventuellement prévenir des attaques *a priori*. En nous basant sur l'hypothèse que l'évolution des pirates informatique a mené à la création d'un environnement où les gens s'entraident et forment des alliances pour améliorer les systèmes qui infectent des usagers, nous pouvons croire qu'il existe des corrélations qui définissent le comportement de chacun de ces groupes. Pour ce faire, nous récoltons une multitude d'informations sur les pages distribuant des virus puis, à l'aide de différentes techniques d'analyse et d'exploration de données, nous tentons de trouver des corrélations d'informations qui définiront des techniques typiques lors d'une attaque.

1.3 Question de recherche

La question de recherche entourant le projet est la suivante : est-il possible, en élargissant la portée des données analysées, de corréler l'information récoltée lors d'une attaque qui nous aidera à identifier des comportements typiques pour prévenir de futures attaques ? Pour cette étude, les attaques sont spécifiquement celles provenant de kits d'exploitation. Nous avons déterminé différents types d'informations que nous voulons récolter pour en extraire des grappes. Afin de répondre à la question de recherche, nous en sommes venus à trois différentes sous-questions de recherche plus spécifiques :

1. Est-ce que le lieu de la personne qui visite la page a un impact sur le comportement de l'infection ?
2. Quelle est la durée de vie moyenne d'une page web qui déploie une attaque ?
3. Est-ce qu'un système de liste noire existe pour optimiser les infections ?

Ces sous-questions nous permettront de déterminer s'il y a beaucoup de variété dans le contenu d'exploitation par rapport au virus et au kit d'exploitation qui sont utilisés. De plus, nous pourrons déterminer si l'infrastructure d'exploitation est différente d'un groupe à un autre.

1.4 Hypothèse

L'hypothèse principale de notre recherche est la suivante : il existe des corrélations entre les mécanismes de distribution de virus et le binaire en soi. La connaissance de ces relations nous permettra de déterminer différents comportements qui nous aideront à prévenir certaines attaques. En effet, nous savons déjà qu'il existe des corrélations du binaire qui permettent d'identifier des comportements distincts d'une attaque à une autre. Il est donc raisonnable de croire que ces corrélations restent valides lorsque nous élargissons la portée des données que nous utilisons et que nous tenons compte des mécanismes de déploiement. De plus, avec l'arrivée et la popularité grandissante des kits d'exploitation et des écosystèmes dans lesquels plusieurs groupes de pirates s'aident mutuellement, nous avons une grande variété de données et nous avons donc une grande probabilité de détecter des corrélations, s'il y en a.

1.5 Méthodologie

Pour répondre à nos hypothèses, nous avons développé un système d'exploration de pages web basée sur la virtualisation. En effet, tel que sera expliqué dans le chapitre 3, notre plateforme, intitulée *PyMalwareAnalyzer* permet de visiter une page web avec un système simulé grâce à VirtualBox[73]. Après l'exploration, la trace réseau ainsi que les données du moniteur de processus de *Windows*[65] sont analysées et nous permettent de générer une multitude d'informations sur ce qui s'est produit sur la machine lors de la visite de la page. Puisque nous utilisons la virtualisation, la configuration de la machine est extrêmement flexible ce qui nous permet d'analyser les comportements pour différents types de machines. Les informations récoltées sont analysées *a posteriori* pour tenter de répondre aux hypothèses que nous avons posées.

CHAPITRE 2

État de l'art

2.1 Détection par signature

Présentement, la majorité des logiciels antivirus disponibles sur le marché utilisent les signatures numériques pour détecter des infections. Une signature numérique représente une partie distincte d'un exécutable qui est utilisée pour repérer le virus. Typiquement, cela peut être une séquence binaire qui se retrouve dans le code ou une séquence d'instructions. L'exécutable est désassemblé à l'aide d'outils de rétro-ingénierie puis des ingénieurs tentent manuellement de déterminer des signatures permettant d'identifier le virus. Cette signature est alors enregistrée dans une base de données qui est alors distribuée aux usagers. La détection se fait en retrouvant la signature parmi les fichiers qui sont examinés.

Lorsqu'un usager possédant un logiciel d'antivirus reçoit un nouveau fichier exécutable sur sa machine, celui-ci est envoyé à l'engin de l'antivirus pour être analysé. Normalement, les engins tentent de détecter si le fichier est compressé, dans le cas échéant, les antivirus utilisent des méthodes intégrées pour effectuer la décompression. L'analyse se fait par la suite sur le fichier décompressé. Dépendant du type de fichier, différents algorithmes de recherche peuvent être utilisés pour accélérer la recherche. L'antivirus tente de trouver des signatures à travers le code désassemblé de l'exécutable. Si une signature est trouvée, le fichier est mis en quarantaine ou effacé puis un rapport d'information est généré et envoyé. Si aucune signature n'est trouvée dans le code, mais qu'il est considéré comme suspect, celui-ci est envoyé à la compagnie d'antivirus pour que des ingénieurs l'analysent.

Le problème majeur de cette technique est qu'il est impossible pour le logiciel d'antivirus de détecter un fichier contaminé qui ne contient aucune signature qui est dans la base de données. La détection de virus par signature n'est donc pas efficace pour les nouveaux virus ou ceux qui sont toujours inconnus. La taille de la base de données est aussi un autre problème puisque chacune des variantes d'un virus représente une nouvelle signature. La taille croît donc de façon exponentielle. Ceci est d'autant plus vrai avec les nouvelles techniques d'obscurcissement et de déploiement rapide utilisées par les gens qui développent des virus. Il est aussi évident que l'utilisation d'un analyste pour la classification n'est pas la méthode la plus efficace. Malgré que celui-ci possède beaucoup d'outils lui permettant de rapidement

classifier de nouveaux exécutables, il reste que l'automatisation de ce processus est beaucoup plus efficace. C'est pourquoi plusieurs recherches tentent de déterminer des méthodes de classification automatique en analysant différentes propriétés lors d'une infection. Il est important de spécifier que maintenant, les compagnies d'antivirus offrent des solutions complètes de protection qui font une analyse beaucoup plus approfondie. Différentes techniques utilisées seront expliquées dans les chapitres suivants.

2.2 Analyse et classification de binaires

La technique la plus répandue pour améliorer la détection de virus est bien entendu la classification de binaires. Il est crucial pour améliorer les détections ainsi que pour mieux comprendre le comportement des développeurs de virus de déterminer des méthodes de classification des binaires. Historiquement, les compagnies d'antivirus utilisaient des signatures numériques pour détecter les binaires. Cette technique, malgré qu'elle soit efficace, n'est plus appropriée avec les techniques de développement rapide d'aujourd'hui, le nombre croissant de nouveaux virus, mais surtout elle ne permet que des détections *a posteriori*. Ce domaine de recherche est donc constamment en évolution et plusieurs techniques furent développées pour améliorer les taux de détection. Nous pouvons généralement classer ces techniques en deux grandes familles : la classification d'attributs statiques et la classification d'attributs dynamiques ou comportementaux. Dans le cas de la classification statique, les chercheurs désassemblent le code du virus pour en extraire des informations pertinentes pour le classifier. Pour la classification dynamique, le virus est exécuté dans un environnement contrôlé où les traces d'exécutions sont utilisées pour représenter le binaire.

2.3 Analyse statique de binaires

Comme la plupart des virus sont distribués sous forme de binaire, l'analyse du code et l'extraction de caractéristiques est donc la base pour classifier des virus. Les compagnies d'antivirus utilisent aussi beaucoup de techniques d'analyse statique pour améliorer leur détection. L'analyse statique d'un binaire s'effectue sur les caractéristiques du code de l'exécutable. Que ce soit des blocs spécifiques de données, le flux de contrôle ou des données, les fonctions ou les appels aux interfaces de programmation (API) pour n'en mentionner que quelques-unes, chacune de ces caractéristiques peut être utilisée pour automatiser la classification d'un binaire. Un des avantages majeurs de la classification statique est que celle-ci permet une analyse quasi complète du code. En effet, comme le programme en entier est analysé, il est possible de corréler le code avec les données du programme. Par contre, le désavantage majeur repose sur le fait que les pirates utilisent plusieurs méthodes d'obscurcissement ou des protecteurs

pour rendre le code décompilé difficile à analyser. Plusieurs recherches se concentrent sur ces techniques pour automatiser la classification de virus, quelques méthodes seront expliquées ci-bas.

2.3.1 Classification sémantique

Dans [12], les auteurs prennent quelques binaires et démontrent qu'en modifiant le code, les antivirus ne détectent plus de signature et ne reconnaissent donc pas le virus. En effet, ils ont développé un outil d'obscurcissement permettant de déjouer les antivirus commerciaux. Quatre techniques sont utilisés pour effectuer les modifications au code, soit :

- L'insertion de code mort
- La transposition de code
- La réassignation de registres
- La substitution d'instruction

L'insertion de code mort consiste à ajouter des instructions qui ne changent en rien le programme (tel que des *NOP*). Pour transposer le code, il suffit de rajouter des branchements pour pouvoir modifier le flux du programme. La réassignation de registres se fait en trouvant un registre non utilisé pour remplacer ceux qui sont utilisés. Finalement, grâce à la diversité d'opérations disponibles pour les architectures Intel, il est possible de modifier une opération par une autre équivalente. Tout ceci permet de modifier la signature d'un binaire. En se basant sur ceci, ils ont créé un outil qui analyse le code d'un exécutable en créant un graphe de flux de contrôle. Ils ont généré un automate qui représente un code malveillant générique et avec lequel ils comparent le code de l'exécutable. Ceci leur permet de déterminer si un code est malicieux. Leur outil, nommé SAFE (*static analyzer for executables*) permet de détecter des virus modifiés qui ne sont pas normalement détectés par les antivirus. Cette méthode se base sur la sémantique du virus pour déterminer si son comportement est malveillant.

Plusieurs différentes techniques de détection par sémantique ont été développées. Par exemple, [14] présente une technique de détection en se basant sur la sémantique d'un programme. Typiquement, un modèle comportemental d'un virus est défini et l'outil tente de le retrouver en analysant la sémantique du binaire. L'outil est plus robuste aux techniques d'obscurcissement, mais il reste assez faible lorsqu'une substitution d'instruction est utilisée. De plus, il prend environ une minute pour analyser un exécutable, ce qui ne le rend pas très pratique pour des fins commerciales.

Kruegel et coll.[57] a développé un système semblable pour détecter les outils de dissimulation d'activité dans le noyau de Linux ou Solaris. Ils utilisent aussi les graphes de flux

de contrôle pour comprendre la sémantique de l'exécutable ELF décompilé. Le problème est un peu différent puisque dans le cas des outils de dissimulation d'activité, le virus doit faire des appels systèmes qu'un module ne devrait pas nécessairement faire. Ils utilisent donc ceci pour pouvoir déterminer la validité d'un module. La technique fonctionne bien, par contre sans l'intégration du système dans le noyau de Linux ou Solaris, il n'est pas d'une très grande utilité.

Finalement, [53] présente une nouvelle technique avec l'utilisation d'une logique de calcul en arbre (CTL). Ils ont développé une logique qui considère les prédicats (CTPL) pour la création de leur arbre. Ils développent un modèle de code malicieux avec leur nouvelle logique (CTPL), puis ils tentent de retrouver ce modèle dans un exécutable analysé.

2.3.2 Classification par comparaison de chaînes de caractères

Une autre méthode couramment utilisée pour une analyse statique de binaires est la comparaison de chaînes de caractères. Un concept fréquemment utilisé pour améliorer la détection par comparaison est la distance de Levenshtein. Cet algorithme, aussi nommé distance d'édition, permet de calculer le nombre minimal d'opérations qui sont nécessaires pour modifier une chaîne de caractères de façon à en obtenir une autre[89]. Gheorghescu[35] sépare un binaire en plusieurs blocs de code d'environ 12 à 14 octets de données. Un bloc de code est défini par une séquence continue d'instructions qui ne contient aucun branchement ou cible de branchement. Grâce à ces blocs, il forme le graphe de flux de contrôle de l'exécutable. L'auteur fait abstraction des protecteurs ou des appels de librairie puisque différentes techniques sont déjà utilisées et offrent de bons résultats. Pour faire l'approximation de la similarité entre deux binaires, trois techniques sont présentées :

- Le calcul de la distance d'édition
- L'index inversé
- Les filtres Bloom[9]

La première technique utilise l'algorithme de distance d'édition pour déterminer la distance entre deux blocs de code. La distance est représentée en octets qui diffèrent entre deux blocs. L'index inversé est une méthode fréquemment utilisée dans les moteurs de recherche de mots. Cette technique consiste à insérer chacun des symboles ou mots d'un document dans une base de données et augmenter un champ qui compte le nombre d'occurrences dans le document. Ces deux techniques présentent cependant des problèmes notamment le temps de calcul dans le cas de la distance d'édition et le nombre d'écritures disque dans le cas de l'index inversé. L'auteur utilise donc les filtres de Bloom qui sont une structure de données probabiliste et faible en espace disque permettant de vérifier si un élément fait partie d'un ensemble. Des

tests sur un ensemble de 4000 binaires malveillants aléatoirement choisis démontrent que non seulement la classification grâce à la comparaison de chaînes de caractères est une technique viable, mais elle est aussi efficace lorsque les bonnes techniques de calcul sont utilisées.

Tian et coll.[92] a effectué une étude comparative de différents algorithmes de comparaison de chaînes de caractères pour automatiser la classification de binaires. Malgré que leur technique fonctionne seulement sur les binaires non protégés et non obscurcis, ils extraient toutes les chaînes de caractères de plus de trois octets d'un ensemble d'environ 1400 binaires de 11 familles différentes. Ils créent un ensemble de toutes les chaînes de caractères se trouvant dans les binaires, puis les algorithmes séparent les binaires en familles en cherchant les chaînes dans les modules de ceux-ci. Une famille est représentée par une liste de modules. Pour accélérer la recherche, un ensemble d'entrainement est généré pour chacune des familles ce qui fait en sorte que seulement les modules utiles d'un binaire seront utilisés pour la recherche. Les 5 algorithmes de classifications comparés sont :

- Machine à vecteurs de support
- Classification naïve bayésienne
- L'arbre de décision
- Forêt d'arbres décisionnels
- Basé sur un exemple

Avec une technique d'accélération nommée *AdaBoost*[33], ils ont réussi à améliorer les performances de 4 des 5 algorithmes de classification et ils réussissent à classer correctement dans leur famille 97% des binaires avec l'algorithme de forêt d'arbres décisionnels. Ils concluent en comparant leurs résultats avec 4 autres études[93, 2, 82, 87] démontrant l'efficacité de leur technique. Islam et coll.[45] présente une technique de classification qui combine le travail de [93] qui se base sur la taille des fonctions ainsi que sur [92] qui compare les différentes chaînes de caractères dans l'exécutable. Une analyse de quelques algorithmes de comparaison est effectuée comme dans [92] et les résultats démontrent qu'en combinant les deux méthodes de classification, plus de 98.8% des binaires sont correctement classifiés.

2.3.3 Classification par appels de fonctions et interfaces de programmation(API)

Une dernière technique étudiée pour analyser statiquement les binaires dans le but de les classifier se base sur le graphe d'appels de fonctions et d'utilisation de librairies et d'interfaces de programmation (API). Un peu comme avec l'analyse sémantique, cette technique s'appuie sur le fait que tous les virus d'une même famille devraient avoir un graphe d'appels similaire. Kinable et coll.[52] présente une technique permettant de regrouper des familles de virus en se basant sur les graphes d'appels de fonctions. Tout d'abord, il explique que la technique la

plus efficace pour déterminer des similitudes entre deux exécutables est d'utiliser la distance d'édition minimale (GED). Plusieurs algorithmes de parcours de graphe sont analysés pour déterminer le plus efficace. Un algorithme polynomial est alors utilisé pour calculer la distance minimale d'édition qui est définie par trois fonctions de coûts :

- Coût des sommets
- Coût des arêtes
- Coût du re-étiquetage

La fonction des sommets représente le nombre de sommets insérés ou enlevés entre deux graphes. Dans le cas du coût des arêtes, cela représente toutes les arêtes équivalentes et finalement la fonction du coût de re-étiquetage désigne les appels de fonctions externes ou internes qui diffèrent. Par la suite, deux algorithmes de regroupement sont comparés puis les résultats démontrent que l'algorithme DBSCAN[27] réussi à grouper correctement les familles de binaires d'ensembles de 194, 675 et 1050 échantillons.

IMDS (Intelligent Malware Detection System)[110] classifie les binaires en examinant les différents appels aux interfaces de programmation (API) de plus de 29 000 exécutables. Ils ont développé un algorithme d'exploration de données basé sur l'association orientée objet. Ils comparent leurs résultats avec différents logiciels antivirus populaires et avec d'autres algorithmes de classification. Leurs résultats sont dans tous les cas supérieurs et leur algorithme a même été implémenté dans le logiciel antivirus de la compagnie KingSoft[54].

Eureka[88], un cadre d'application, permet de décompiler des exécutables compressés ou obscurcis afin de faire une analyse. Grâce à un pilote de bas niveau qui permet de suivre les appels systèmes, l'application filtre les appels de l'identifiant du processus (PID) de l'exécutable. En se basant sur le fait que le code sera décompressé lorsque le processus meurt ou crée un nouveau processus, ils peuvent retrouver le code décompressé en mémoire. Ils utilisent aussi une analyse des bi-grammes pour résoudre les binaires non résolus avec la méthode heuristique. Finalement, ils ont développé une technique permettant de résoudre les appels à des interfaces de programmation (API) obscurcies. La plateforme a réussi à décompresser (entièrement dans la plupart des cas) plus de 90% des exécutables analysés et la résolution des interfaces de programmation apporte beaucoup d'information pertinente pour une analyse statique.

2.3.4 Limites de l'analyse statique

Bien que l'analyse statique puisse donner des résultats intéressants pour automatiser la classification de binaires, il reste que les gens qui développent des virus tentent constamment

de déjouer ces techniques. Dès qu'une nouvelle méthode est présentée, il suffit que les pirates trouvent un moyen de contrer celle-ci pour la rendre inutile. Par ailleurs, l'analyse statique est souvent effectuée sur le contenu d'un binaire. Ceci peut représenter un problème dans le cas des virus polymorphes ou métamorphiques. De plus, l'approximation est souvent utilisée pour analyser statiquement un virus et, dans la plupart des cas, les approximations sont plutôt conservatrices pour éviter les faux positifs. Moser et coll.[67] présente différentes approches permettant d'échapper aux détections de plusieurs des méthodes qui ont été présentées ci-haut. Comme quelques-unes sont utilisées dans des logiciels antivirus commerciaux, nous aurions pu décider d'en inclure dans notre système. Par contre, tel que sera présenté dans le chapitre 4, nous n'avons pas inclus d'algorithme de classification statique dans notre système final. Dans notre cas, ce qui s'apparente le plus à notre solution est la classification dynamique de binaires qui sera revue dans la section suivante.

2.4 Analyse dynamique de binaires

Comme mentionné dans la section précédente, un des problèmes majeurs de la classification par analyse statique est la difficulté de reconnaitre du code qui a été obscurci par un protecteur. La méthode la plus répandue pour contrer ce problème est d'analyser le virus lors de son exécution et non pas le contenu de celui-ci. Il est évident que lorsque le binaire est exécuté, qu'il soit obscurci ou non n'a plus d'importance. Comme il sera exposé dans le chapitre 4, notre système fait, en quelque sorte, une classification dynamique de virus. Il existe plusieurs façons de classifier les virus dynamiquement, un peu comme pour l'analyse statique, les appels de fonctions, les appels aux interfaces de programmation peuvent être analysés, mais d'autres éléments tels que la trace réseau ou les différents appels systèmes sont plus spécifiques à une classification dynamique.

2.4.1 Classification comportementale

Une des façons les plus courantes de classifier des binaires dynamiquement est d'analyser le comportement du virus lors de son exécution. Ceci peut être fait de plusieurs façons. Par exemple, Christoderescu et coll.[13] présente une méthode qui considère un virus comme une boîte noire. En effet, une machine se trouvant dans un environnement limité exécute un virus et toutes les interactions avec le système d'exploitation sont alors récoltées. Plus précisément, ils considèrent les appels système comme base pour ensuite construire un graphe. Avec la trace d'exécution, un graphe de dépendance d'appels est généré. Deux appels sont considérés comme dépendants lorsque les valeurs de leurs arguments sont les mêmes et que le type d'information est semblable. Par exemple, un appel **RegCreateKeyA** sera dépendant

de l'appel subséquent **RegDeleteValueA** si leurs arguments concordent. Pour augmenter l'efficacité de leur algorithme d'exploration de données, ils réduisent les graphes en agrégeant les appels similaires, ils vérifient les dépendances des sous-chaînes de caractères et observent les autoréférencements. Ils appliquent, par la suite, un algorithme d'exploration de données sur les sous-graphes réduits. Grâce à leur technique, ils peuvent détecter des comportements malicieux dans une trace d'exécution, et ce sans avoir de connaissance *a priori* du binaire. L'algorithme a été testé sur un ensemble de 16 binaires et la plupart des comportements jugés malveillants par des experts en analyse de binaires de chez Symantec ont été identifiés par cette technique.

Les auteurs dans [98] présentent une technique similaire pour classifier des familles de virus. Un système de virtualisation via *Wine* est utilisé pour exécuter des virus *Win32*. Pour éviter les détections, aucun débogueur ou outil de surveillance n'est utilisé puisque ceux-ci sont facilement détectables par des virus. L'environnement contrôlé exécute le binaire pour 10 secondes et les messages d'exécution de *Wine* sont extraits pour l'analyse. Un peu comme[13], les appels d'interface de programmation sont utilisés pour créer un graphe de ressemblance. La fréquence relative des appels est compilée puis la distance de Hellinger[103] est utilisée pour générer les distances associées des appels. Leur système permet de créer un arbre phylogénétique[105] qui permet de facilement classifier les familles de virus ainsi que les virus qui ne sont pas encore classés. Leurs tests sont exécutés sur 104 binaires avec environ 7% d'incohérences dans la classification.

Kolbitsch et coll.[55] présentent un système assez intéressant permettant d'analyser et de classifier des virus en temps réel avec une réduction de performance assez minime. Grâce à une version améliorée de leur système d'analyse nommé *Anubis*[6, 4], l'auteur génère un graphe d'appels système. Leur méthode leur permet d'extraire plusieurs informations supplémentaires telles que la mémoire virtuelle à chaque appel ou les dépendances de données à chaque appel. En fait, les dépendances de données sont extrêmement importantes puisqu'elles permettent de déterminer si la sortie d'un appel système sera l'entrée d'un autre appel. Grâce à la richesse des informations extraites, il est possible de détecter des comportements malicieux lors d'une exécution. Ils utilisent aussi le découpage de programme pour pouvoir déterminer différentes fonctions qui pourraient être malicieuses. Pour les détections, lorsque leur scanneur détecte un appel intéressant (potentiellement malveillant), une analyse de dépendance est effectuée pour déterminer si celui-ci a un comportement malicieux. Leurs résultats sont assez intéressants, dans le cas de l'ensemble d'entrainement, sur 300 binaires analysés, 279 sont détectés. Puis, sur un restant de 263 binaires analysés, les résultats sont

plus faibles et obtiennent un taux de détection d'environ 64%. Par contre, sur les 263 binaires, 108 exemplaires représentent des variantes inconnues et le taux de détection est d'environ 23% alors que sur les 155 autres exemplaires connus, le taux est de 92%. Ceci démontre que malgré une faible détection pour des variantes inconnues, il y a tout de même presque 25% qui seront détectés. De plus, leur système n'est pas très lourd en performances avec une baisse de performances de moins de 10% sauf dans des cas extrêmes tels que la compilation. La technique a été poussée un peu plus loin dans [32] qui a développé un outil appelé *HOLMES* pour la détection et classification de comportement malicieux. En effet, leur méthode utilise différentes techniques déjà prouvées[82, 2, 112] pour améliorer leur travail précédent[13]. Les résultats présentés sont prometteurs avec un taux de détection de 86% sur plus de 1000 exécutables, comparativement à 40-60% pour les détecteurs comportementaux commerciaux, 55% pour les logiciels antivirus et 64% de [55]. Le temps de détection le plus long pour un nouveau virus était de 12-48 heures, ce qui est tout de même beaucoup plus efficace comparativement au temps de détection de nouveau virus qui serait de 54 jours selon[22].

2.4.2 Analyse par trace réseau

Bien évidemment, avec la constante évolution de l'informatique, les pirates trouvent de nouvelles façons d'exploiter des failles pour pouvoir infecter un maximum de personnes possible. En effet, John et coll.[47] ont analysé le trafic de polluriels de l'Université de Washington pour déterminer que plus de 75% du trafic malveillant est généré par 6 réseaux de robots de virus connus. Le fait le plus intéressant est que la majorité du trafic généré par ces virus se fait sous forme de requêtes *HTTP*. C'est ce qui a inspiré Perdisci et coll.[75] à analyser le comportement des virus par les traces réseau qu'ils génèrent. En effet, les communications réseau sont un bon moyen de faire abstraction du comportement de ce qui est exécuté sur la machine infectée. Leur méthode démontre que, malgré que différentes variantes d'un virus possèdent des comportements assez distincts, leurs traces réseau sont souvent très similaires. À l'aide de différentes méthodes de regroupement, des grappes sont formées et permettent de détecter des virus que les logiciels antivirus ne détectent pas. Une première étape de regroupement à granularité grossière trie les exemplaires de virus par leur nombre de requêtes *HTTP*, par le nombre de requêtes *GET* et *POST,* ou par la longueur des *URL* pour n'en nommer que quelques-uns. Des grappes sont alors formées et une deuxième étape de regroupement est effectuée. Cette fois, pour avoir de plus petites grappes, les similarités entre la structure des requêtes *HTTP* sont analysées à l'aide de la mesure de distances. Après cette étape, le centre des différentes grappes est évalué puis les grappes avec des centres près sont réunies pour créer une signature du comportement réseau. Leur expérimentation s'effectue sur une période de 6 mois où ils ont capté plus de 25,000 virus distincts. À la fin de chaque mois,

des signatures sont créées pour détecter des virus futurs. Avec un ensemble d'entrainement de 3 mois, le taux de détection est d'environ 60%. Pour la détection de virus qui ne sont pas encore reconnus par les antivirus, leur taux de détection est d'environ 55-60%. Malgré que leur méthode produise beaucoup de faux positifs et qu'il faille donc étalonner les règles générées, il est tout de même évident que l'analyse du trafic réseau de virus produit un bon taux de détection. Gu et coll.[38] a lui aussi développé une méthode similaire pour la détection de réseau de robots. L'étude est plus spécifique, car elle est orientée sur la détection en entreprise de machines compromises grâce aux communications qu'elles effectuent au centre de contrôle par IRC, HTTP ou même de pair-à-pair.
Une étude intéressante de Rossow et coll.[83] présente une analyse de traces réseau de virus effectuée sur une période d'une année. En effet, plus de 100,000 exemplaires de virus et de leurs comportements réseau ont été étudiés permettant d'en connaitre davantage sur les moyens typiques de communication des binaires. Sur l'ensemble, environ 45% des virus exécutés ont généré des activités réseau, totalisant plus de 200 GB de données. Une exécution dure en moyenne 7.8 jours avec des temps allant de 6 heures jusqu'à 120 jours. Il est intéressant de noter que 92.3% des traces utilisent le protocole DNS, 58.6% utilisent HTTP et 8% utilisent IRC. L'étude est très détaillée sur le type de paquets qui sont envoyés ou reçus, analysant les paramètres des différentes requêtes GET ou POST de HTTP, par exemple. Malgré qu'aucun algorithme de classification ne permette de distinguer le trafic bénin du trafic malveillant, plusieurs détails intéressants sur les communications générées par les virus sont présentés. Des algorithmes de regroupement tels que celui présenté par [10] pourraient facilement aider à mieux comprendre le comportement des virus.

2.4.3 Analyse avec virtualisation

Tout comme pour notre étude, analyser des binaires à l'aide de la virtualisation est assez commun. Jiang et coll.[46] a effectué une étude présentant une technique novatrice pour détecter de nouveaux virus ou des outils de dissimulation d'activité. Leur méthode permet de détecter « hors de la boîte » les virus, ce qui veut dire que malgré que l'exécution du virus s'effectue dans un environnement virtuel, la détection de celui-ci est effectuée sur la machine hôte. Grâce aux images de la mémoire ainsi que des données du disque qui sont fournies par les outils de virtualisation, *VMWatcher*, leur système, permet de détecter en temps réel une infection. Le système est assez efficace et permet de dissimuler les outils de détection, par contre, comme il sera mentionné plus tard, les virus qui tentent de détecter s'ils s'exécutent dans un environnement virtuel sont plutôt rares et l'effort d'une telle méthode ne permet que de détecter ceux-ci.

Puisqu'un environnement de virtualisation permet de restreindre la portée du virus exécuté et qu'il permet aussi de capturer une multitude d'informations pertinentes, son utilisation pour l'analyse de binaires est presque idéal. Par exemple, Willems et coll.[108] présentent un système de virtualisation qui permet de générer des rapports détaillés de différents aspects lors de l'exécution d'un virus. Un peu comme le système que nous avons développé, leur approche est plutôt orientée sur l'analyse que sur la classification. En effet, avec l'utilisation de *CWSandbox*, un instrument de virtualisation, un rapport détaillé est généré et décrit :

- Les fichiers créés ou modifiés par le binaire
- Les modifications aux registres Windows
- Les librairies dynamiques (DLL) chargées
- Les secteurs de la mémoire virtuelle accédés
- Les processus créés
- Les connexions réseau effectuées
- Toute autre information pertinente comme les accès aux secteurs protégés, les services installés ou les pilotes de noyau.

La méthode est basée sur l'accrochage des appels aux interfaces de programmation ainsi que toutes les injections de librairies dynamiques. La machine virtuelle est initialisée et les opérations nécessaires pour capturer l'information d'exécution sont exécutées. Ensuite, le virus est exécuté et la machine attend que le processus meure ou sinon, elle s'arrête après un certain délai. L'analyse est ensuite faite sur toutes les informations de l'exécution. Une analyse de plus 6000 binaires durant 5 mois a permis de déterminer plusieurs caractéristiques intéressantes sur les virus, ce qui simplifie grandement le travail d'analyse.

Bayer et coll.[5] a aussi réalisé une étude pour analyser des binaires à l'aide de la virtualisation. *Anubis*, leur système de virtualisation, permet à des gens d'envoyer des fichiers exécutables, qui seront exécutés à l'intérieur d'une machine virtuelle fonctionnant avec *Qemu*[7]. Leur système est toujours en ligne[97] et continue de compiler des statistiques sur les binaires reçus. Il est maintenant aussi possible d'analyser des exécutables pour Android. L'article rapporte des statistiques sur les binaires analysées depuis une période d'environ 2 années. Près de 1 million d'exécutables uniques (basé sur la fonction de hachage MD5) ont été récoltés et des grappes comportementales ont été générées. Plusieurs statistiques intéressantes telles que les protecteurs les plus couramment utilisés, les comportements typiques des virus ou les activités réseau sont générés. Dans le cas des comportements, l'étude rentre en détail sur ce que les virus font assez fréquemment, comme ajouter une clé au registre Windows pour s'exécuter au démarrage par exemple. Les activités réseau aussi sont analysées plus en détail et comme les autres études, il est évident que beaucoup de virus effectuent des requêtes sur Internet par un centre de commandes. *HTTP* reste encore le protocole le plus

utilisé et il semblerait que l'utilisation de *IRC* soit plutôt en déclin selon leurs résultats. D'autres statistiques intéressantes présentent les activités typiques d'un réseau de robots et démontrent que *IRC* reste, tout de même, le protocole le plus populaire pour ce type de virus. Finalement, une analyse des virus, qui font des détections de virtualisation, démontre qu'il n'est pas très fréquent pour les virus de faire de telles vérifications. Parmi leur ensemble de virus, moins de 100 binaires feraient une analyse pour détecter des outils de vérification. Nous pouvons donc en déduire que cela reste un comportement marginal et que les résultats n'en sont que très peu biaisés.

2.4.4 Limites de l'analyse dynamique

Tout comme l'analyse statique, tenter d'analyser un virus lors de son exécution possède aussi certaines limitations. Egele et coll.[24] présentent une enquête sur les techniques d'analyse dynamique de binaire, qui résume les travaux majeurs qui ont été réalisés sur ce sujet. Comme il a été présenté dans la section 2.4, il existe plusieurs méthodes pour analyser dynamiquement un exécutable malveillant, par contre, il existe certaines limitations. En effet, un des problèmes majeurs est la vue limitée qu'une analyse d'exécution offre. Effectuer une exécution est assez demandant en terme de temps, mais ne permet pas d'analyser tous les différents chemins d'exécution. L'analyse statique est plus complète sur ce point puisque le code désassemblé contient tous les chemins d'exécution. Un autre problème est le fait que l'exécution ne s'effectue que pendant une certaine période de temps. Ceci limite les résultats puisqu'il n'est pas assuré que, durant cette période, le binaire exécutera du code malveillant. Par exemple, un simple délai peut être implémenté forçant le virus à ne s'exécuter qu'à une date précise. Sinon, nous pouvons penser à un exécutable qui se connecte à un réseau de robots et qui n'effectue rien tant qu'il ne reçoit pas de commande. Il est aussi plutôt difficile de simuler l'interaction d'un usager, un virus qui requiert une certaine interaction ne sera donc pas nécessairement exécuté comme il se doit. Finalement, l'analyse dynamique est aussi beaucoup plus gourmande en terme de temps et de ressources.

2.5 Analyse et classification web

L'Internet est sans aucun doute le moyen le plus simple et efficace pour des développeurs de virus de distribuer leurs logiciels malveillants. Plusieurs recherches se penchent sur l'analyse et la classification des pages web qui distribuent des virus pour tenter de mieux comprendre les techniques utilisées par les pirates. Dans [68], un système d'explorateur de pages a permis de classifier un très grand nombre de pages qui distribuent des binaires malveillants. Malgré que l'étude date de 2005, les résultats sont assez intéressants. Leur système d'explorateur

est assez similaire à ce que nous avons développé, mais l'étude est plutôt orientée sur le téléchargement d'exécutable puis l'analyse de ceux-ci. Leur méthode exécute le binaire puis, s'il y a lieu, installe le programme. Ensuite, grâce au logiciel AdAware[59], ils effectuent une analyse de la machine virtuelle pour vérifier s'il y a eu infection. Plusieurs statistiques intéressantes sont produites, mais comme leur méthode dépend de l'efficacité de détection du logiciel antivirus, il est plutôt difficile pour eux de détecter de nouveaux virus. Leurs URL proviennent de la base de données de Heritrix[41]. De ceci, plus de 2500 noms de domaine sont extraits comme point de départ pour naviguer. Les domaines sont de plusieurs catégories différentes, des sites pour adultes, de jeux, pour enfants, pour des fonds d'écrans, pour n'en nommer que quelques-uns. Plus de 18 millions d'URL sont visitées chaque mois et, selon leurs statistiques, des 19% de sites contenant des exécutables, 13,4% sont infectés le premier mois et 5.5% le deuxième mois. Une partie de leur étude se concentre aussi sur les téléchargements forcés, ce qui s'approche beaucoup de notre étude. Pour détecter une infection, leurs heuristiques sont :

- Création d'un processus
- Activité dans le système de fichiers (autre que des endroits tels que la cache)
- Un processus suspect écrit un fichier
- Activité dans les registres
- Une interruption inattendue dans le fureteur ou le système d'exploitation

Selon leurs observations, en mai 2005, environ 6% des URL font du téléchargement forcé alors que moins de 1% en font en octobre. L'étude présente aussi les différences d'infection lorsqu'Internet Explorer[63] est utilisé versus l'utilisation de Firefox[69].

Cette technique d'exploration de page s'apparente beaucoup à ce qui se fait avec les serveurs pots de miel. Ces serveurs tentent d'attirer les bots d'un réseau ou tout type d'attaque pour pouvoir ensuite les analyser. Dans le cas présent, nous parlons plutôt d'un client pot de miel, parfois nommé *honeyclient*. Le but est de faire visiter le web à un client pour que celui-ci se fasse attaquer ou infecter. Il existe deux grandes familles de clients, ceux avec une grande interaction et ceux avec une petite interaction. Les clients avec une grande interaction, ce qui s'apparente le plus à notre système, simulent des systèmes complets sans limitation fonctionnelle. Ils sont extrêmement efficaces pour trouver de nouvelles attaques, par contre, puisqu'ils sont plus lourds à exécuter, ils sont moins performants. Les clients à petite interaction utilisent plutôt des versions réduites de clients, un peu comme les explorateurs de pages, ce qui les rend extrêmement rapides, mais moins efficaces puisque les pirates peuvent les détecter assez facilement. Vaagland et coll.[95] décrivent les différents types de clients pot de miel et énumèrent les implémentations les plus populaires ainsi que leurs différences.

Comme il a été présenté auparavant, l'évolution de l'Internet et le nombre croissant d'usagers ont permis aux pirates informatiques de développer de nouveaux vecteurs d'infection. En effet, depuis déjà quelques années, une méthode très populaire pour distribuer des virus est utilisée et permet des infections à l'insu presque total de l'usager. Ces infections par téléchargement forcé permettent aux pirates d'infecter leurs victimes en exploitant des failles dans les modules d'aide au fureteur ou dans des modules d'extension par exemple. Egele et coll.[23] présentent les différentes méthodes utilisées par les pirates pour initier des attaques par téléchargement forcé. Que ce soit par la mauvaise utilisation d'une interface de programmation, l'exploitation d'une vulnérabilité dans le fureteur ou ses modules d'extensions, les pirates réussissent à infecter les usagers sans aucune interaction avec celui-ci.

Provos et coll.[77] ont réalisé une étude pour classifier des URL qui distribuent des virus avec des téléchargements forcés. Leur source d'URL provient des explorateurs de pages de Google, ce qui leur a permis de visiter plus de 4.5 millions d'URL différents. Le système qu'ils utilisent fait une analyse pour déterminer une base d'URL susceptibles de contenir du code malveillant, puis une exploration des pages permet ensuite de déterminer s'ils distribuent réellement des virus. Pour chaque URL visitée, un pointage est établi selon des heuristiques prédéfinies telles que des activités dans le système de fichiers, des processus créés, les résultats de logiciel antivirus, etc. L'étude analyse les différentes méthodes utilisées par les pirates pour injecter du contenu malicieux à l'intérieur d'un site. Les quatre méthodes les plus communes sont l'exploitation de failles dans le serveur ou son code, le contenu des usagers, les publicités ainsi que les gadgets logiciels. Une fois que les pirates ont accès au contenu de la page web par une de ces quatre méthodes, il suffit d'injecter du code exploitant des failles du fureteur, des modules d'aide ou des modules d'extension. Souvent, les pirates feront des vérifications pour vérifier si l'usager navigue avec Internet Explorer ou Firefox. Ils peuvent aussi faire des inspections sur la machine virtuelle de Java pour déterminer s'il existe des failles potentielles sur celle-ci. Une autre technique est de tromper un usager en lui faisant télécharger un module d'extension pour pouvoir visionner la page alors que l'usager télécharge plutôt un virus. De toutes les URL visitées, environ 10% distribuent des virus (environ 450,000 URL) parmi lesquels environ 300,000 sont des chevaux de Troie, alors qu'environ 18,000 sont des publiciels et 35,000 distribuent des binaires non classifiés. Ces classifications sont basées sur les résultats des logiciels antivirus, ils ne sont donc qu'indicatifs. Une partie de l'étude porte sur l'évolution des virus dans le temps. Selon leurs statistiques, les URL distribuant des logiciels malveillants changent le binaire qu'ils distribuent à une fréquence allant de quelques heures jusqu'à quelques jours.

Mavrommatis et coll.[61] présentent une recherche qui est, en quelque sorte, une continuité du travail présenté ci-dessus [77]. Grâce aux mêmes outils, ils ont pu classifier plus de 65 millions de pages web parmi lesquelles plus de 3 millions sont suspectes, plus de 3 millions sont malveillantes et utiliseraient plus de 180,000 sites d'infection. L'exploration s'étend sur une période de 1 an et plusieurs statistiques intéressantes confirment ce que le travail précédent démontrait. En effet, un nombre assez élevé des URL distribuant du contenu malicieux utilise des versions contenant des failles de sécurité pour leur serveur ou même pour leur langage de script (p.ex. PHP). Les attaques par des campagnes publicitaires sont aussi extrêmement utilisées et elles sont assez difficiles à prévenir puisque celles-ci contiennent à 50% plus de 6 redirections différentes. Il est donc assez difficile pour un site avec de la publicité de retracer d'où provient l'infection qui contamine ses usagers. L'étude est très détaillée sur la provenance des sites d'infections, par exemple, près de 70% des pages infectées proviennent d'une certaine façon de Chine. Grâce aux différentes traces réseau, ils ont pu cerner près de 10,000 sites de distribution de virus, parmi lesquels 90% proviennent d'une seule adresse IP. Finalement, des inspections sur la machine virtuelle d'exploration permettent d'observer que lors d'une infection, plusieurs centaines de processus sont lancés pour installer des modules d'aide au fureteur, modifier des préférences du fureteur, modifier des paramètres de sécurité du système d'exploitation ou pour modifier les registres pour que le virus s'exécute au démarrage.

Zhang et coll.[111] ont aussi réalisé une étude sur la détection d'URL potentiellement malicieuses qui contiennent des téléchargements forcés. Leur système de détection de téléchargements forcés, nommé *ARROW*, utilise des traces HTTP en entrée. Chaque URL est visitée puis le contenu dynamique (c.-à-d. JavaScript) qui s'y trouve est exécuté. Puisque leur système ne contient aucune automatisation d'interactions humaines, une URL est signalée comme suspicieuse si un exécutable est écrit sur le système de fichier. Avec les URL potentiellement malicieuses, une analyse est effectuée pour déterminer les serveurs centraux qui distribuent des virus. Une signature est générée avec l'aide d'expressions régulières, ce qui permet de déterminer des modèles généraux pour identifier des pages qui distribuent des binaires malveillants. Leur système a visité près de 3.5 milliards d'URL, parmi ceux-ci, près de 15,000 grappes d'adresses Internet et d'IP sont détectées contenant environ 7000 réseaux de distribution de virus. Près d'une centaine de ces réseaux utilisent un ou plusieurs serveurs centraux. Le taux de faux positifs général est de 0.0019%, mais le système gère plus difficilement les cas où des serveurs légitimes sont impliqués augmentant le taux jusqu'à 0.4%. Un des gros problèmes de l'étude est la simplicité de l'heuristique de détection de pages mali-

cieuses. Comme il sera démontré par notre étude, les pirates peuvent utiliser d'autres types de fichier pour infecter un usager ce qui ne serait pas détecté par *ARROW*. De plus, l'utilisation de signature, comme pour la détection de binaires malveillants, permet aux pirates de contourner le problème en générant des versions polymorphiques de leurs binaires ou des URL.

Finalement, Grier et coll.[36] ont réalisé une étude sur le phénomène du MaaS qu'il nomme plutôt « *Exploit-as-a-Service* » dans leur article. Ceci s'apparente beaucoup à notre recherche, dans le sens où nos sources principales d'infection proviennent de kits d'exploitation tels qu'ils sont présentés dans l'article. Il existe deux types de clients potentiels pour les kits d'exploitation typiques. Tout d'abord, il existe les gens qui paient pour rediriger du trafic vers le kit d'exploitation. Ces gens reçoivent un certain montant d'argent basé sur le trafic qu'ils génèrent. Il existe aussi les gens qui développent de nouveaux exploits, ceux-ci vendent les nouveaux binaires aux gens qui s'occupent du kit d'exploitation et se font payer pour chaque installation. Avec ce paradigme, le trafic est généré automatiquement par les gens qui paient pour avoir accès au kit, puis le kit est constamment mis à jour avec des nouveaux binaires qui sont fournis par des développeurs. Leur étude diffère de la nôtre dans leur méthodologie. Pour leur analyse, 6 sources différentes de binaires sont utilisées :

- Google, grâce à leur « *Safe Browsing infrastructure* » [61]
- MDL[60], une liste publique d'URL contenant des binaires
- Sandnet[83]
- Des pièces jointes de courriels
- Torrents
- *ATLAS*[70]

Grâce à ceux-ci, plus de 60,000 binaires sont captés, exécutés et analysés dans un environnement virtuel contrôlé fonctionnant avec *GQ honeyfarm*[56]. Pour limiter les impacts externes et pour imiter le comportement des adresses que le virus contacte, plusieurs services internes sont utilisés pour répondre aux requêtes DNS, HTTP et SMTP. Pour classifier les différents binaires, plusieurs techniques de classification comportementale sont utilisées :

- Domaines visités
- Arguments HTTP
- Modification du système
- Prise d'écran

Une partie de l'étude est spécifique aux attaques par téléchargement forcé. Grâce à leur technique de regroupement, il est possible d'identifier les différents kits d'exploitation utilisés. Avec l'analyse des URL visitées, un regroupement a permis de déterminer que *Blackhole*[91, 43] est le kit le plus répandu avec environ 28% des infections totales. La dernière partie

de leur étude porte sur la popularité et la durée de tous les domaines utilisés dans une infection par téléchargement forcé. Avec la base de données des recherches DNS passives du Security Information Exchange[28], contenant en moyenne 726 millions de requêtes DNS quotidiennement, il a été possible de déterminer que la durée moyenne de vie des domaines utilisés dans une infection par téléchargement forcé ne reste en ligne que 2.5 heures.

CHAPITRE 3

Solution proposée

Comme il a été discuté dans les sections précédentes, l'écosystème dans lequel les pirates informatiques travaillent de nos jours a beaucoup évolué. Bien qu'il soit difficile de bien comprendre le fonctionnement interne de cet environnement, nous suspectons que plusieurs groupes collaborent et s'entraident pour pouvoir maximiser les infections, et par conséquent les revenus de chacun. Pour ce faire, nous avons développé un système d'exploration de pages qui récolte une multitude d'informations. Selon notre hypothèse, la variété de données que nous récoltons pourra nous aider à trouver des corrélations qui nous permettront de mieux comprendre la complexité du fonctionnement de ce milieu. Par fonctionnement, nous entendons déterminer les différents groupes, les différentes contributions qu'il peut y avoir entre ceux-ci, les méthodes et outils utilisés, tout ceci dans le but de mieux prévenir des attaques. Ce chapitre explique en détail le système que nous avons utilisé et ses différentes composantes. En premier lieu, nous allons brièvement décrire le système globalement puis nous discuterons du planificateur qui se charge d'organiser les explorations avec les différentes machines ainsi que l'IP à utiliser. Par la suite, nous expliquerons les données que nous récoltons et que nous traitons avec notre module de traitement. Finalement, nous allons décrire le module final qui se charge d'analyser les données pour en sortir des informations pertinentes.

3.1 PyMalwareAnalyzer

Notre système, développé en python[31], est séparé en cinq modules principaux :

- Le récolteur d'URL ;
- Le planificateur ;
- L'explorateur ;
- Le module de traitement ;
- Le module de statistiques.

Ce qui sépare notre système de tous les clients pot de miel est la complexité et la diversité de l'information que nous récoltons. De plus, nous nous distinguons beaucoup puisque les informations proviennent de plusieurs sources, ce qui nous permet de bien définir l'environnement dans lequel l'infection se produit. Le schéma de la figure 3.1 explique les différents modules avec les informations qu'ils prennent en entrée puis les données qu'ils envoient au prochain module. Les rectangles représentent les différents modules développés et les paral-

lélogrammes symbolisent les différentes entrées que les modules reçoivent. Pour simplifier, le schéma ne représente que les principaux modules et actions. Un schéma plus complet de notre système, un diagramme de la base de données avec les éléments importants de chacune des tables se trouve en annexe.

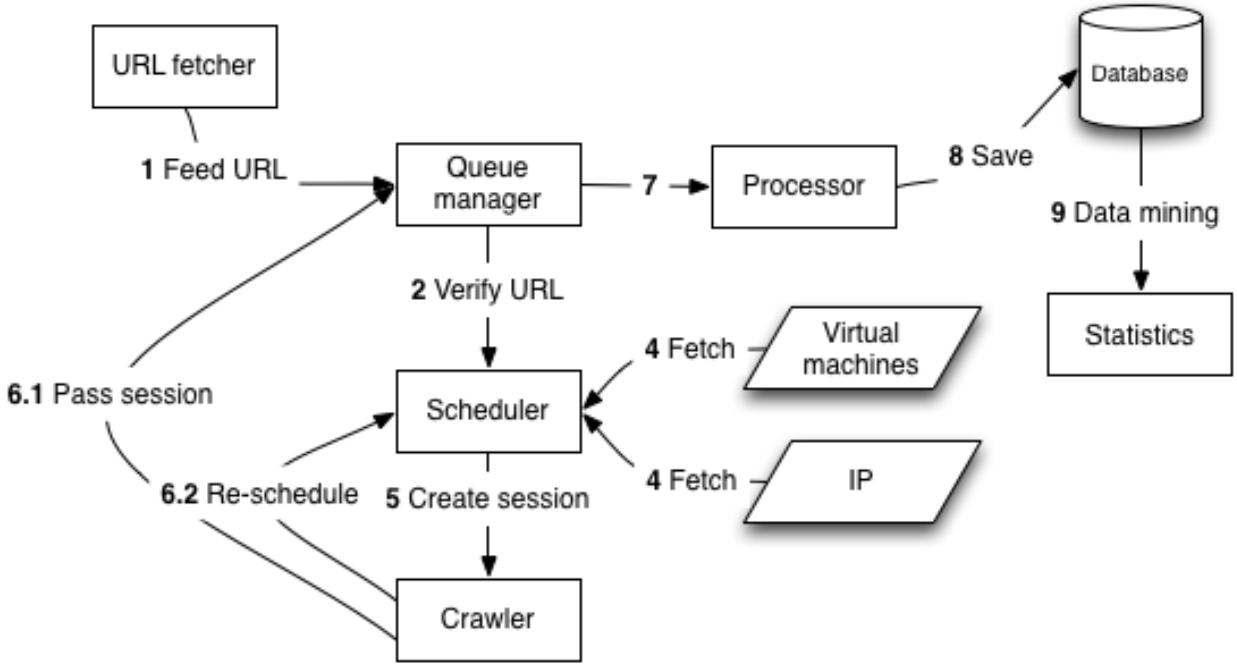


Figure 3.1 Diagramme des différents modules du système réalisé.

Sommairement, notre système récolte des URL provenant de différentes sources puis les envoie au gestionnaire de files. Le gestionnaire de files prend la plus récente URL et l'envoie au planificateur qui lui s'organise pour choisir la bonne machine virtuelle avec un IP. Une fois ceci déterminé, l'explorateur se charge d'exécuter l'exploration de la page, puis de passer les données récoltées, sous forme de fichier PCAP et du moniteur de processus, au module processeur. Celui-ci trie les informations puis les sauvegarde dans notre base de données. *A posteriori*, notre module de statistiques permet d'extraire les informations intéressantes de façon à les présenter sous forme de résultats concrets. Les prochaines sections expliqueront en détail ce que chacun des modules fait.

3.2 La récolte d'URL

L'efficacité de notre système repose sur la qualité des URL que nous fournissons à l'explorateur. C'est pourquoi nous avons développé un système qui permet de récolter des URL

de différentes sources pour ensuite les envoyer au gestionnaire de files pour que celui-ci crée la tâche d'exploration. Le module ne fait qu'amasser des URL en passant par des requêtes HTTP ou bien par un courriel que nous avons créé pour recevoir des URL malveillantes. Les différentes sources seront expliquées dans le chapitre 4.

3.3 Gestionnaire de files

Le gestionnaire de files est un module essentiel, mais relativement simple à comprendre. Sommairement, il gère les files d'attente qui fournissent les URL à explorer et les sessions à analyser. Les URL proviennent de différentes sources puis sont insérées dans une file à priorité avec le temps comme facteur de priorité. La valeur du temps est représentée en secondes et est désignée par le temps lorsque l'URL est insérée dans la file avec ou sans délai. En effet, comme il sera expliqué dans la section 4.1, pour réaliser certains tests, les URL peuvent être explorées plusieurs fois, et parfois elles sont explorées après un certain délai. Avant d'envoyer l'URL à explorer au module de planification, le temps (en secondes) est comparé au temps présent, et dans le cas où le temps associé à une URL est après le temps actuel, le gestionnaire de files attendra la différence de temps :

Avec : t_a = Temps actuel, t_i = Temps d'insertion et t_e = Temps d'exploration

$$t_e = t_a - t_i$$

$$Si : t_e > 0$$

$$Attendre(t_e)$$

Dans le cas des données de session à envoyer au module processeur, après une exploration, le module d'exploration envoie la session au gestionnaire, qui lui l'insère dans la file de sessions à analyser. Sans avoir de priorité, la file des tâches à analyser permet de les distribuer pour, éventuellement, faire du traitement en parallèle, comme discuté dans la section des travaux futurs.

3.4 Planificateur

Le module de planification s'occupe d'établir avec quelle machine l'exploration se fait ainsi que l'adresse qu'elle prendra. L'URL reçue en entrée permet de déterminer les différentes sessions qui ont déjà été effectuées sur celle-ci, pour ensuite établir la bonne combinaison de machine virtuelle et adresse pour la prochaine exploration. La méthodologie que nous avons

utilisée pour déterminer ceci est expliquée plus en détail dans la section 4.1, cette section se veut plutôt une explication du fonctionnement de ce module. Le schéma 3.2 montre la vue globale du planificateur.

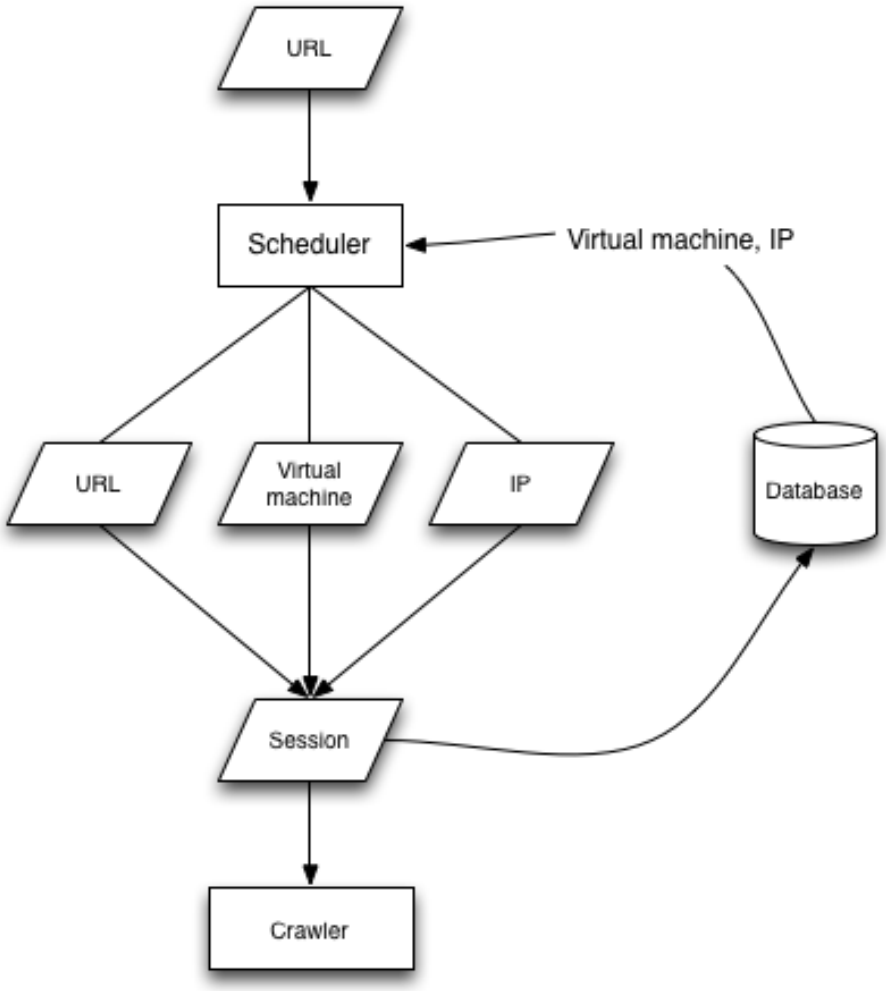


Figure 3.2 Schéma du module de planification.

3.4.1 Validation de l'URL

Initialement, le planificateur reçoit une URL qui lui provient du gestionnaire de file qui a, au préalable, été validée par une recherche DNS. Quoique cette technique pour valider si une URL est toujours en ligne comporte des erreurs, elle reste la plus efficace pour éviter des détections potentielles par les pirates. En effet, au tout début, notre système effectuait une requête avec l'outil *WGet*[99], ce qui nous donnait beaucoup plus d'informations pour déterminer si le site web était encore en ligne. Comme *WGet* simule une requête HTTP vers le site

web, nous pouvions analyser la réponse pour déterminer si, par exemple, celle-ci donnait une erreur. Le problème majeur de cette méthode est qu'en effectuant une requête avec *WGet*, nous compromettons notre adresse IP, ce qui pourrait biaiser les résultats dans le cas ou des règles de liste noire sont utilisées. C'est pourquoi l'utilisation de recherche DNS nous évite de nous faire détecter, mais ne nous fournit pas d'information *a priori* sur la réponse du serveur.

Avec une URL que nous savons valide, le planificateur vérifie d'abord si celle-ci a déjà été explorée au préalable. Ceci lui permet de déterminer, grâce aux algorithmes qui seront expliqués dans la section 4.1, la prochaine machine virtuelle ainsi que l'adresse IP pour la prochaine exploration, selon le test qui est effectué. Avec ces données, le planificateur crée un objet de session qui sera le lien à travers le système pour retracer tout ce qui a été effectué lors de l'exploration. Une fois l'exploration terminée, le module d'exploration peut repasser par le module de planification pour une prochaine exploration d'une même URL. Encore une fois, selon le test qui est exécuté, l'URL est remise en file d'attente ou peut tout simplement se faire explorer successivement avec différents attributs (adresse IP, machine virtuelle).

3.4.2 Objet session

Notre système contient une base de données très diversifiée en information, et pour les lier les unes aux autres, nous utilisons un objet que nous avons défini comme une session. L'objet session est le pilier de notre algorithme d'exploration de données puisqu'il permet d'extraire toutes les informations recueillies lors d'une visite de page par notre système. Le tableau 3.1 énumère les différents attributs d'une session.

Tableau 3.1 Objet de type session

Session		
Nom	Type	Description
URL	Clé étrangère	L'URL à explorer
time	Date	Date de l'exploration
path	Chaine de caractères	Chemin vers les fichiers issus de l'exploration
virtual_machine	Clé étrangère	La machine virtuelle qui a fait l'exploration
IP	Clé étrangère	L'adresse IP utilisée pour l'exploration

L'objet n'est pas très complexe en soi, mais il est important de le définir puisqu'il est un concept important qui est utilisé dans plusieurs des prochains modules.

3.5 Explorateur

Le module d'exploration est en quelque sorte le cœur de notre système, il reçoit les informations de session qui lui permettent de mettre en place le RPV puis de sélectionner la machine virtuelle pour l'exploration. La figure 3.3 explique ce que le module fait exactement lors d'une exploration de page avec les données qui en sortent.

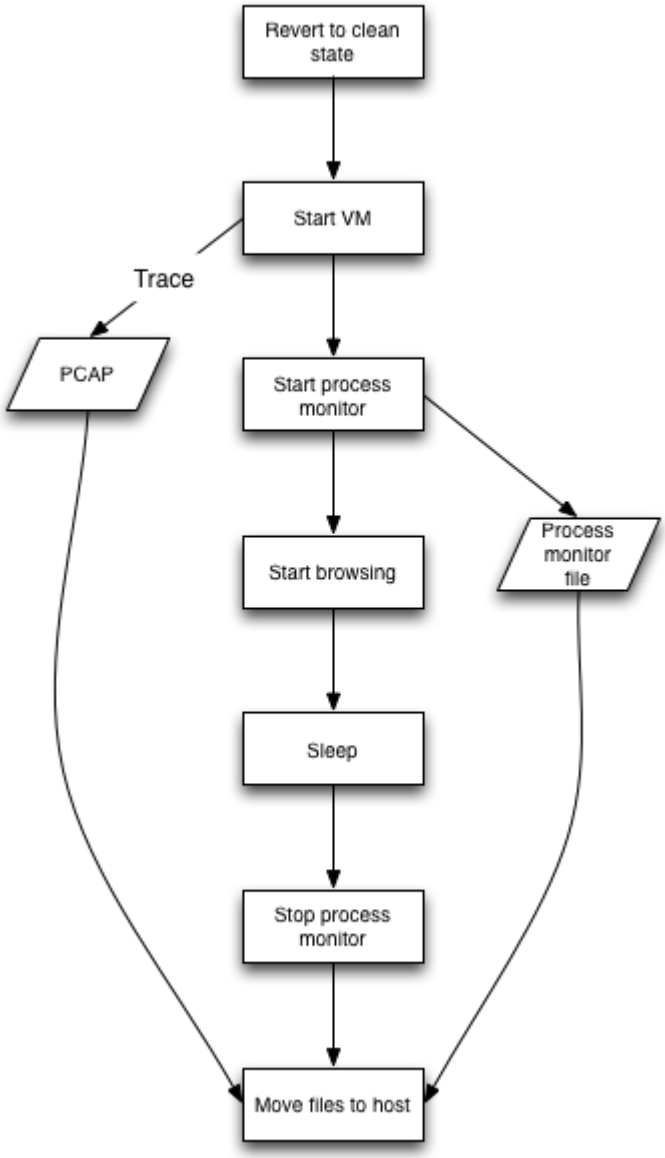


Figure 3.3 Graphe de flot de contrôle du module d'exploration.

Le processus de visite de page est simple et intuitif, le but ici étant de simuler le plus précisément possible le comportement d'un humain qui visiterait ce site. Nous avons un

instantané de la machine dans un état que nous savons propre, c'est-à-dire que la machine n'est pas infectée et est vulnérable aux différentes attaques. Une fois cette image rétablie, un script démarre le moniteur de processus puis lance un fureteur vers l'URL à visiter. Par la suite, la machine s'endort pour un temps donné pour que, s'il y a lieu, l'attaque se mette en place. Finalement, un script arrête le moniteur de processus et déplace le fichier de sortie vers un endroit sur la machine hôte où se trouve aussi le fichier de trace réseau. Ces deux fichiers sont ensuite transférés au module de traitement en passant par le gestionnaire de files. Comme il a été mentionné dans la section 3.3, il existe une file de tâches d'exploration et de traitement.

3.6 Module de traitement de données

Bien que le système nécessite une bonne infrastructure logicielle, tout ceci est réalisé dans le but d'extraire un maximum d'informations pour pouvoir ensuite les analyser. Les données sortantes du module d'exploration sont brutes et doivent être traitées minimalement pour alléger les algorithmes d'exploration de données. Notre recherche tente de déterminer s'il existe différentes corrélations entre les virus et l'information contextuelle que nous récoltons lors de l'exploration. Cette section explique les différentes informations qui sont traitées après une exploration de page ainsi que les raisons pour lesquelles celles-ci sont pertinentes pour les corrélations. Comme il sera discuté dans la section suivante, ces informations permettent, entre autres, de déterminer si une exploration entraine une infection de la machine virtuelle. Ces données sont aussi directement liées avec les résultats de notre recherche. Les prochaines sous-sections seront divisées de façon à décrire les informations récoltées de la machine virtuelle qui fait l'exploration, les données récoltées sur la page visitée *a priori* (DNS, *WhoIs*, domaine) puis finalement, les informations issues de l'exploration *a posteriori*

3.6.1 Informations de la machine virtuelle

Avant même que notre système ne fasse une exploration, il existe déjà plusieurs informations que nous récoltons pour établir des corrélations. Comme il sera expliqué à la section 4.1, les informations récoltées de la machine virtuelle permettent de définir un visiteur spécifique puis de déterminer si le comportement de la page visitée est modifié par une variation de celui-ci. Le diagramme de la figure 3.4 résume les informations que notre système récolte à chaque exploration de page sur la machine virtuelle utilisée. Nous avons regroupé ces données en 4 ensembles distincts, chacun possédant des attributs qui peuvent faire varier le comportement du site visité.

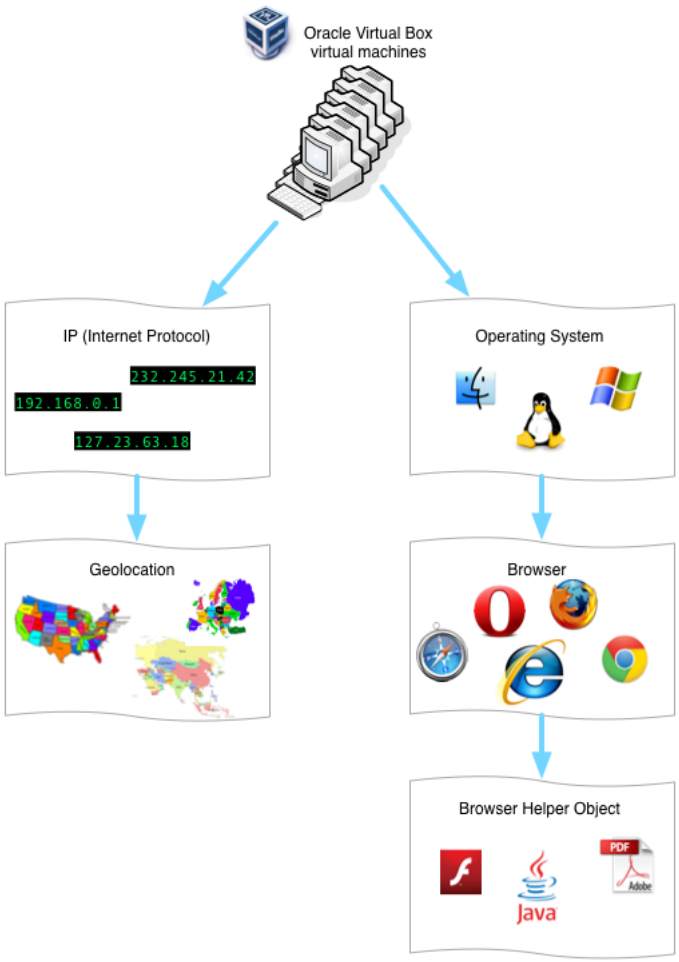


Figure 3.4 Diagramme des informations récoltées des machines virtuelles.

Système d'exploitation

Tout d'abord, notre système sauvegarde des informations sur le système d'exploitation qui est utilisé pour visiter la page web. Grâce à l'utilisation de *Virtual Box*, il est possible de faire l'exploration d'une même page web avec différentes configurations de système d'exploitation. Les informations spécifiques au système d'exploitation qui sont sauvegardées sont :

1. Nom
2. Version
3. Paquet de services (*Windows* **seulement**)

Le nom du système d'exploitation est l'information la plus globale que nous avons, elle représente la famille de celui-ci. Globalement, les trois grandes familles de système d'exploitation qui sont ciblées par des attaques, mais qui ne se limitent pas à celles-ci, sont :

- *Microsoft Windows*
- *Mac OS*
- *Linux*

Les virus sont toujours développés pour un type spécifique de système d'exploitation, mais souvent ils exploitent des failles de versions spécifiques de ceux-ci. Dans le cas des kits d'exploitation, il est assez fréquent que ceux-ci fassent une vérification sur le système d'exploitation pour pouvoir distribuer le bon virus à leur potentielle victime. C'est donc pour cette raison que cette information est intéressante pour bien comprendre le contexte dans lequel le virus peut infecter des gens. Par exemple, les différentes versions de *Windows* sont : XP, Vista, 7, 8, alors que pour *Mac OS* ou *Linux* ce sont vraiment les numéros de version qui sont utilisés, respectivement : 10.0.1, 10.8.4, 10.5.8 pour *Mac OS* ou 3.1, 3.10, 2.6.34 dans le cas de *Linux* pour n'en énumérer que quelques-unes. Finalement, le paquet de services est une information propre au système d'exploitation de *Microsoft*. En effet, pour une même version du système, Microsoft développe parfois des paquets de services, par exemple, SP1, SP2 et SP3 pour la version XP de *Windows*. Ces paquets représentent une information pertinente puisque souvent, ils comprennent des correctifs de sécurité qui peuvent éviter certaines infections[66].

Fureteur

Le deuxième ensemble de données que notre système sauvegarde se rapporte aux informations sur le fureteur utilisé pour faire l'exploration. Chacun des fureteurs offre un certain système de protection contre des infections, ce qui altère le comportement des pages explorées. *NSS Labs*[80] a effectué une étude sur les différents fureteurs et la sécurité qu'ils offrent. Cette étude démontre bien que certains navigateurs sont très agressifs dans leur protection et bloquent beaucoup de pages. Cependant, nous enlevons toute sécurité des fureteurs pour ne pas biaiser les résultats. Les données sur les versions des navigateurs sont tout de même intéressantes puisque, entre autres, les kits d'exploitation utilisent le fureteur et sa version pour décider du virus à envoyer à l'usager, tel que discuté dans [48, 20]. Les trois données que nous récoltons sur les fureteurs sont donc :

1. Nom
2. Version
3. Modules d'aide au fureteur (sous-ensemble)

Tout comme pour les systèmes d'exploitation, le nom représente la famille du navigateur. Encore une fois, il existe plusieurs familles de fureteurs, les 5 plus grands compétiteurs dans ce domaine sont :

- *Google Chrome*
- *Microsoft Internet Explorer*
- *Mozilla Firefox*
- *Apple Safari*
- *Opera*

La version est elle aussi une information importante puisque des correctifs de faille de sécurité sont souvent apportés de version en version. Par contre, la donnée la plus importante sur les navigateurs est celle sur les modules d'aide qui sont installés avec ceux-ci.

Objet d'aide au fureteur

Un objet d'aide au fureteur est un module de librairie qui agit comme module d'extension pour le navigateur en ajoutant des fonctionnalités à celui-ci[102]. Bien qu'aujourd'hui ces modules sont très communs et souvent nécessaires pour profiter pleinement de certains sites, ils restent tout de même une assez grande source de risque. Les pirates exploitent des failles sur ces objets, ce qui leur donne plus d'accès à la machine et donc plus de liberté pour infecter les usagers. Les informations que nous récoltons sur les objets d'aide sont :

1. Nom
2. Version

Encore une fois, il existe une multitude de modules d'aide, mais il est évident que d'un point de vue d'amplitude d'infections et de potentiel économique, les pirates tentent de trouver des failles dans les objets d'aide les plus populaires. Les trois modules que nous avons déterminés étant les plus susceptibles de contenir des failles sont :

- *Oracle Java*
- *Adobe Reader (PDF)*
- *Adobe Flash*

Tout comme pour les autres données, le numéro de version de l'objet d'aide est très important puisque les pirates se servent directement de cette information pour envoyer le virus qui exploitera la faille de la version spécifiquement. Ces trois données nous permettent de simuler un usager typique des pages web et grâce à la virtualisation, nous pouvons modifier les paramètres facilement.

IP et géolocalisation

Parmi les hypothèses que nous avons émises, nous pensons que la localisation de l'usager a une incidence sur son infection et que les pirates utilisent une liste noire pour ne pas infecter les mêmes usagers à plusieurs reprises. Pour valider ceci, nous avons donc développé un

système pour pouvoir modifier l'adresse IP de la machine virtuelle. SecDev[37] nous a fourni différentes adresses IP provenant de régions différentes sur la planète auxquelles nous nous connections par un RPV. Ce système nous donne une certaine flexibilité, mais permet aussi de limiter l'étendue des infections de nos machines virtuelles. En effet, comme l'exploration d'une page web peut provoquer une infection de la machine et que ceci peut générer beaucoup de trafic indésirable, il était important d'utiliser un système réduisant le risque d'impact sur le réseau de l'école.

3.6.2 Informations de la page *a priori*

Les informations récoltées de la machine virtuelle nous permettent de mieux comprendre le contexte d'un client qui se ferait infecter, les informations que nous récoltons sur la page permettent de définir le contexte de distribution des virus. Plus spécifiquement, les données que nous obtenons avant l'exploration sont importantes pour les décisions des tests, comme il sera discuté à la section 4.1. Pour l'analyse de nos résultats, les données intéressantes proviennent du module de résolution du nom de domaine. Avant même d'explorer la page, notre système fait une requête DNS pour déterminer les différentes adresses IP associées à l'URL à visiter. Combiné avec les informations récoltées *a posteriori*, ceci permet de bien cerner le contexte où les virus infectent leurs victimes.

3.6.3 Informations issues de l'exploration *a posteriori*

Les données que nous ramassons après l'exploration sont surement les plus riches et celles qui seront les plus utiles pour répondre à nos hypothèses. Le module de traitement extrait et sauvegarde les informations de la trace réseau et du fichier du moniteur de processus.

Données réseau

La trace réseau qui est capturée durant toute l'exploration permet de sauvegarder toutes les informations qui sont transmises entre le client et l'extérieur. Le traitement réseau se fait en trois étapes :

- Traitement du domaine
- Traitement du fichier PCAP
- Traitement des fichiers

Le traitement de domaines est plutôt simple : il consiste à résoudre l'URL en adresse IP(s), puis de récolter les informations *WhoIs*[100] ainsi que le *WhoIs* de l'équipe *Cymru*[21]. Les différences entre les deux types sont décrites aux tableaux 3.2 et 3.3

Tableau 3.2 Informations récoltées des requêtes *WhoIs*

WhoIs		
Nom	Type	Description
name	Chaine de caractères	Nom de l'hôte
date_registration	Date	Date d'enregistrement
date_expiration	Date	Date d'expiration
emails	Chaine de caractères	Courriel(s) associés
registrar	Chaine de caractères	Nom(s) du registraire
status	Chaine de caractères	Status
name_servers	Chaine de caractères	Serveur(s) de nom (DNS)

Tableau 3.3 Informations récoltées des requêtes *WhoIs Cymru*

WhoIs Cymru		
Nom	Type	Description
IP	IP	IP associé au domaine
as_num	Chaine de caractères	Numéro du système autonome
as_name	Chaine de caractères	Nom du système autonome
bgp_prefix	Chaine de caractères	Prefix du BGP (classe d'IP)
country_code	Chaine de caractères	Code du pays
date_allocated	Date	Date d'allocation
hostname	Chaine de caractères	Nom de l'hôte

Les données récoltées du *WhoIs Cymru* sont plutôt orientées vers les systèmes autonomes. Un système autonome est un ensemble de réseaux faisant partie d'Internet et possédant une politique de routage interne cohérente. Le routage entre deux systèmes autonomes est défini par un préfix BGP[101]. Ces informations sont donc plus globales, car elles représentent le réseau qui permet de se rendre à l'hôte. Dans le cas des informations *WhoIs*, elles désignent plutôt ce qui touche directement au domaine et à celui qui l'enregistre.

Une fois les domaines résolus, le traitement du fichier PCAP permet d'extraire toutes les requêtes qui ont été faites durant l'exploration de la page. Globalement, l'ensemble des requêtes est trié en 4 grandes familles :

- Trafic UDP
- Trafic TCP
- Trafic DNS
- Trafic HTTP

Comme le trafic UDP et TCP sont plus larges, nous ne sauvegardons que les IP de la destination ainsi que de la source de la connexion et les ports associés à ceux-ci. Par contre,

puisque le trafic DNS et HTTP possèdent plus d'informations, nous avons séparé les requêtes des réponses puis nous sauvegardons les données tel que représenté aux tableaux 3.4, 3.5, 3.6, 3.7 et 3.8.

Tableau 3.4 Informations récoltées des requêtes *WhoIs Cymru*

Requête DNS		
Nom	Type	Description
name	Chaine de caractères	Nom du serveur
type	Nombre	Type d'enregistrement DNS
dns_class	Nombre	Classe DNS

Tableau 3.5 Informations récoltées des réponses DNS

Réponse DNS		
Nom	Type	Description
name	Chaine de caractères	Nom du serveur
type	Nombre	Type d'enregistrement DNS
dns_class	Nombre	Classe DNS
ttl	Nombre	Nombre de seconde(s) durant lesquelles l'enregistrement est valide
res	Chaine de caractères	Nom de l'hôte résolu

Tableau 3.6 Informations récoltées des requêtes HTTP

Requête HTTP		
Nom	Type	Description
method	Chaine de caractères	Méthode de la requête (GET, POST, HEAD, etc.)
host	Chaine de caractères	Nom de l'hôte
uri	Chaine de caractères	Arborescence de la requête
referer	Chaine de caractères	Nom de la référence vers la requête. Typiquement cela est représenté par une URL
user_agent	Chaine de caractères	Description de l'agent (fureteur) faisant la requête

3.7 Module de statistiques

Le dernier module de notre système est celui qui nous permet d'extraire les informations puis de les comparer de façon à obtenir des résultats concrets qui répondent à nos hypothèses. L'objectif de ce module est d'explorer les données récoltées pour trouver des corrélations possibles entre celles-ci. C'est le seul module qui ne fonctionne pas en ligne avec le reste du

Tableau 3.7 Informations récoltées des requêtes HTTP

Réponse HTTP		
Nom	Type	Description
date	Date	Date de la réponse
expires	Date	Date lorsque la requête n'est plus valide (pour la cache)
last_modified	Date	Date de dernières modifications
connection	Chaine de caractères	Type de connexion préféré de l'agent (ex. : *keep-alive*, *close*)
server	Chaine de caractères	Nom du serveur (ex. *Apache*, *nginx*, *ECS*)
content_type	Chaine de caractères	Le nom du type MIME du contenu
status	Chaine de caractères	Le code du statut HTTP (ex. 200, 404, 500)

Tableau 3.8 Informations récoltées des fichiers HTTP

Fichier HTTP		
Nom	Type	Description
path	Chaine de caractères	Chemin du fichier sur le disque
filename	Chaine de caractères	Nom du fichier
sha1	Chaine de caractères	Hachage SHA1 du fichier
sha256	Chaine de caractères	Hachage SHA256 du fichier
md5	Chaine de caractères	Hachage MD5 du fichier

système. En effet, pour effectuer l'exploration de données, il n'est pas nécessaire d'analyser les informations en temps réel et, puisque tout est stocké sur le disque dur et la base de données, le module fait l'analyse des données *a posteriori*. Le module ne fait qu'extraire les données intéressantes de façon à les corréler pour sortir des résultats qui répondent aux questions. Le chapitre 5 présente les différentes données que nous avons utilisées avec les résultats qu'elles suggèrent en les comparant.

CHAPITRE 4

Méthodologie

Dans ce chapitre, nous décrivons comment nous utilisons notre système d'exploration de pages dans le but d'augmenter notre connaissance de l'écosystème dans lequel les pirates informatiques exécutent leurs opérations. *PyMalwareAnalyzer* permet d'explorer des pages web avec des machines virtuelles pour tenter d'infecter celles-ci. Lorsqu'une infection a lieu, une analyse des informations récoltées nous permet de trouver des corrélations entre les différentes pages. Dans un premier temps, nous expliquons les différentes sources qui nous alimentent en URL avec leurs avantages et inconvénients. Puis, nous détaillons les métriques utilisées afin d'établir si une infection a eu lieu lors de l'exploration du site web. Par la suite, nous allons discuter de la méthodologie que nous avons développée pour nos différents tests dans l'optique de répondre à nos hypothèses initiales.

4.1 Méthodologie de tests

Le système que nous avons développé est en quelque sorte l'outil que nous utilisons pour construire une base d'informations qui nous permettra de répondre à nos hypothèses, mais aussi de mieux comprendre le milieu dans lequel les pirates collaborent. Pour répondre spécifiquement à nos hypothèses, nous avons donc implémenté une méthodologie pour réaliser différents types de tests. Tout d'abord, nous allons expliquer les différents paramètres de nos tests, puis nous détaillerons les trois tests que nous avons créés dans le but de répondre à nos questions de recherche.

4.1.1 Paramètres de tests

Pour réaliser nos expériences, il faut tout d'abord établir les différents paramètres que nous pouvons modifier pour arriver à nos résultats. En nous basant sur les différents modules qui ont été définis au chapitre 3, nous pouvons définir 7 arguments pour nos tests :

- Système d'exploitation
- Fureteur
- Objets d'aide au fureteur
- Temps d'exploration
- Source d'URL
- Planification d'exploration

– Adresse IP

De tous ces critères, il nous faut décider lesquels nous gardons fixes et lesquels nous faisons varier. Étant donné la nature de nos hypothèses, plusieurs de ces paramètres peuvent rester fixes durant tous nos tests sans biaiser nos résultats. Plus précisément, la configuration de la machine virtuelle (système d'exploitation, fureteur, objets d'aide au fureteur), le temps d'exploration et la source d'URL restent fixes durant tous les tests.

Configuration de la machine virtuelle

La raison pour laquelle la configuration de la machine virtuelle reste fixe durant tous les tests est simplement pour simplifier ceux-ci. Différentes configurations à n'importe quel niveau (système d'exploitation, fureteur, objet d'aide au fureteur) augmentent la richesse de nos données, mais ne nous apportent rien de nouveau pour nos hypothèses. Prenons l'exemple de l'hypothèse que les pirates ne distribuent pas les mêmes virus selon la localisation du client. En partant avec cette prémisse, visiter la même page web avec différentes configurations de machine virtuelle nous apporte certainement une plus grande variété de données, par contre, en fixant le profil de la machine virtuelle, nous pouvons déterminer que pour un type de machine, l'hypothèse est confirmée ou pas. Ceci s'applique à toutes les hypothèses que nous avons formulées dans la section 1.4. Nous pourrions ensuite extrapoler que pour toute autre configuration, les résultats seraient les mêmes. Malgré ceci, modifier les paramètres de la machine virtuelle n'est pas nécessaire pour confirmer nos hypothèses.

Temps d'exploration

Un deuxième critère que nous avons décidé de fixer pour les expérimentations est le temps d'exploration. Comme notre système n'effectue aucune interaction avec la page web, il est plutôt inutile que celui-ci reste pendant une durée trop longue au repos sur le site. Nous avons fixé le temps d'exploration d'une page à 2 minutes 30 secondes durant toutes les expériences. Ce temps permet, selon nous, à l'attaque de se mettre en place et d'infecter la machine qui la visite. Bien que d'avoir un temps fixe d'exploration comporte des compromis, nous jugeons que ceux-ci restent négligeables si nous voulons garder l'efficacité de notre explorateur. En effet, il a déjà été soulevé que quelques virus prenaient un certain temps avant de se manifester (timebomb). Par contre, dans tous les cas, le virus aura été transmis à la victime, et nous pourrions donc le détecter en se basant sur nos critères d'infection tels que définis à la section 4.2. Il se peut aussi que l'attaque n'ait pas le temps de se compléter durant le délai de 2 minutes et 30 secondes pour des raisons de latences de réseau par exemple. Dans ce cas, nous pourrions plus difficilement détecter une infection. Pour garder notre système efficace

et rapide, nous avons décidé que ces cas sont négligeables.

Source d'URL

Pour que notre système soit capable de fonctionner à son plein potentiel et qu'il extrait le maximum d'informations pertinentes des différentes pages visitées, il faut que les URL que nous visitons soient non seulement malicieuses, mais qu'elles infectent la machine facilement. Pour des raisons de simplification, nous avons décidé que notre système d'exploration ne simulerait aucune interaction humaine, ce qui veut dire que l'infection doit s'effectuer automatiquement. Dans cette section, nous expliquons les deux sources qui peuvent nous fournir des URL malicieuses, soit par des listes en ligne ou par un client courriel.

URL par liste en ligne :
Il existe plusieurs organismes qui œuvrent dans le domaine de la sécurité informatique pour récolter des URL potentiellement malveillantes. Malware Domain List[60], VX Vault[96] ou Malware Patrol[18] pour n'en nommer que quelques-unes, sont des listes d'URL malicieuses qui sont constamment mises à jour par la communauté. La plupart sont disponibles gratuitement pour visionnement et représentent donc une source très intéressante pour notre système. La technique d'extraction est très simple, il suffit de recueillir les listes puis, pour chacune d'elle, développer un module de décomposition analytique pour en extraire les URL. Initialement, notre système n'utilisait que les URL fournies par ces listes, par contre, plusieurs problèmes nous ont forcés à utiliser une autre méthode de cueillette d'URL. Tout d'abord, ces listes contiennent des milliers, voire des millions d'entrées qui ne sont plus nécessairement valides. Non seulement, la qualité des URL est discutable mais, dans la plupart des cas, elles sont le point final de la chaîne d'infection. C'est-à-dire que souvent les URL pointent directement sur un fichier à télécharger et à installer par l'usager. Comme il a été mentionné dans le chapitre 3, notre système ne simule pas d'interaction usager, ce qui rend les URL que nous visitons peu intéressantes en terme d'informations récoltées.

URL par un client courriel :
Pour pallier ce problème, nous avons décidé de créer une boîte courriel qui reçoit des URL que nous pouvons ensuite extraire avec un module de décomposition analytique. Bien que le courriel est public et que n'importe qui pourrait envoyer des URL à celui-ci, en ce moment il n'est utilisé que par des membres de la compagnie d'antivirus ESET[25]. Grâce à une entente entre le laboratoire de sécurité des systèmes informatiques de Polytechnique (SecSI) et ESET, nous recevons, de façon horaire, un courriel automatique qui contient une multitude d'URL qui ont potentiellement infecté les usagers de leur système d'antivirus. Cette technique

d'extraction d'URL comporte plusieurs avantages par rapport à l'extraction par les listes en ligne :

- Les URL sont toujours récentes (moins d'une heure)
- L'URL peut pointer n'importe où dans la chaîne d'infection (souvent au début)
- Les URL sont triées par ESET avant de nous être transmises
- Plusieurs différents types d'infection (JavaScript, PDF, kit d'exploitation)

Pour simplifier, et comme la source provenant de ESET était plus que suffisante pour réaliser toutes nos expériences, nous avons limité notre système à s'alimenter uniquement de celle-ci pour fournir des URL à explorer. Le fait d'utiliser uniquement la source de ESET comme base d'URL à visiter biaise un peu nos résultats puisque ce sont seulement des URL que ESET juge comme malicieuses et qui respectent certains critères qui nous sont fournis. Par contre, ceci n'a pas d'impact sur nos résultats puisqu'il n'y a aucune différence si l'URL est jugée comme malveillante selon ESET ou Malware Domain List. Le fait est que les URL qui nous sont données sont potentiellement dangereuses et les critères qu'ils utilisent au préalable pour filtrer les URL ne font que réduire la portée de notre système. Nous pouvons tout de même prouver le concept sur une plus petite source d'URL.

4.1.2 Plan d'expérimentations

Pour valider nos hypothèses, nous avons construit un plan d'expérimentation qui nous a permis de bien diviser le problème en trois expériences distinctes. Chacune des expériences est exécutée sur une période d'une semaine, durant laquelle notre système explore les pages qu'il reçoit par la source d'URL de ESET. Tel qu'il a été mentionné dans la sous-section 4.1.1, certains des paramètres de tests restent fixes durant toutes les expériences pour simplifier celles-ci. Un point important est que la configuration de machine virtuelle que nous utilisons durant nos tests soit vulnérable. Pour nous assurer de ceci, nous avons utilisé un système d'exploitation sans les dernières mises à jour, un fureteur qui permet de visiter des pages web potentiellement malicieuses. Notre choix s'est arrêté sur Firefox, puis d'anciennes versions de tous les objets d'aide au fureteur. Les BHO que nous avons utilisés sont *Java*, *Adobe Reader* et *Flash*, tous avec des versions qui présentaient des vulnérabilités en se basant sur [81] et [16]. Les prochaines sous-sections décrivent les trois différentes expérimentations que nous avons effectuées en expliquant les paramètres qui varient et les raisons pour lesquelles nous effectuons ces tests.

Test sur la géolocalisation

Le premier test que nous effectuons tente de déterminer si les pirates utilisent un algorithme d'ajustement comportemental en se basant sur la localisation des visiteurs de la page infectée. Pour ce faire, nous utilisons pleinement la puissance de notre planificateur qui permet de modifier dynamiquement l'adresse IP de la machine virtuelle. La plage d'adresses IP qui nous est fournie couvre 3 pays dans 3 continents différents, plus précisément : différentes villes aux États-Unis, Londres et Tokyo. Notre hypothèse est motivée par l'utilisation grandissante des rançongiciels qui sont reconnus pour leur utilisation de la position géographique de leur victime pour envoyer un binaire différent, ou du moins, leur afficher un message différent en se basant sur le pays de la personne infectée [85, 84, 34]. Pour confirmer cette hypothèse, il nous faut donc visiter une même page avec des clients de différents endroits.

L'élément le plus important lors de la réalisation de cette expérience est la fonction de choix d'adresse IP qui est utilisée par le planificateur pour déterminer le prochain IP d'exploration. La fonction est relativement simple :

Avec :	$previous\_countries$	Tel que ,
	$random(ips)$	IP aléatoire excluant l'ensemble de pays

$$next\_ip = random(previous\_countries)$$

$$previous\_countries = previous\_countries \cup next\_ip.country$$

À la toute première exploration, l'adresse IP du client est déterminée de façon aléatoire, puis un tableau contenant les pays des anciennes explorations se remplit. Aux explorations suivantes, l'adresse est choisie de façon aléatoire dans un ensemble excluant les adresses qui proviennent de pays qui ont déjà été utilisés. Chacune des explorations est effectuée successivement et sans délai pour s'assurer que le page est toujours en ligne. La raison pour laquelle nous utilisons le pays au lieu de la ville est strictement pour diversifier autant que possible nos résultats. Quoiqu'il soit possible que les pirates distribuent des binaires en se basant sur la ville du visiteur, pour éviter de nous faire détecter et pour simplifier le problème, nous jugeons qu'un pays différent devrait procurer une assez grande diversité dans nos résultats. Si notre hypothèse est infirmée, il sera très clair qu'il y a une différence entre les pages qui sont distribuées ou affichées à l'usager relativement à sa localisation.

Pour déterminer si une session d'exploration diffère d'une autre, il nous faut un algorithme nous permettant de calculer la similitude entre deux sessions. Pour ce faire, nous avons pris

les informations de deux sessions puis nous comparons chaque binôme de sessions qui visite la même URL. La table 4.1 énumère les différents groupes d'information que nous utilisons dans l'algorithme de similarité ainsi que les propriétés que nous omettons du calcul de similitude. Le schéma de la base de données en annexe décrivent les tables et leurs différentes propriétés.

Tableau 4.1 Sous-groupes de données utilisées pour la similarité entre sessions

Nom	Propriété(s) omise(s)
Requête DNS	Aucune
Réponse DNS	Aucune
Connexion UDP	Port de destination et port de source
Connexion TCP	Port de destination et port de source
Requête HTTP	Aucune
Réponse HTTP	date
Fichier HTTP	*path*
Moniteur de processus	Aucune
Fichier binaire	Aucune

Pour la plupart des sous-groupes, toutes les propriétés des objets sont utilisées en omettant évidemment les références vers d'autres objets et les identifiants. Dans le cas des connexions (TCP et UDP), nous ignorons les ports puisque celui entrant est généré aléatoirement pas le système d'exploitation. Pour la réponse HTTP, nous décidons de négliger la date qui est évidemment toujours différente pour deux réponses distinctes. Pour les fichiers HTTP, le chemin est ignoré puisqu'il est absolu et donc, pour deux sessions différentes, il sera toujours différent. Les fonctions de hachage (MD5, SHA1, SHA256) sont beaucoup plus exactes pour évaluer la similitude.

Calcul de similarité : Avec ce groupe d'informations, nous avons développé un algorithme de calcul de distance entre une paire de sessions. Le calcul de similarité se fait sur deux sessions qui ont visité une même URL. En effet, bien qu'il soit possible de calculer la similitude entre n'importe quel binôme de sessions, pour notre étude et nos tests il est plus intéressant de se concentrer sur les similitudes pour une même URL, pour nous indiquer les différences comportementales du site lorsque nous varions la configuration de la machine. L'algorithme de similitude est détaillé en pseudo-code à l'algorithme 1.

Données : tableau1[], tableau2[], diff[]
Résultat : Pourcentage de similarité entre tableau1 et tableau2
Précondition : grandeur(tableau1) > grandeur(tableau2)

pour *x dans tableau1* **faire**
  si *!tableau2.contient(x)* **alors**
    diff.inserer(x)
  fin
fin
retourner *1 - (grandeur(diff)/grandeur(array1))*

Algorithme 1: Algorithme de similitude entre deux sessions

Comme nous pouvons le voir, l'algorithme compare chacun des objets de la liste la plus grosse puis insère chacun des objets différents dans une nouvelle liste. La valeur de retour représente le pourcentage de similitude entre les deux listes. Par exemple, si la première liste contient 20 éléments et la deuxième en contient seulement 10 de la première liste, alors notre algorithme retourne une similarité de 50%. Si nous appliquons l'algorithme aux sessions, pour chacun des binômes de sessions, nous comparons chacun des groupes de données définis dans la table 4.1, puis nous sauvegardons la similitude de chacun d'entre eux.

Prenons l'exemple des connexions TCP pour démontrer le calcul de la similarité. Le tableau 4.2 montre deux tableaux de données TCP à comparer.

Tableau 4.2 Exemples de données TCP à comparer

Premier jeu de données			
IP source	Port source	IP destination	Port destination
10.12.13.14	300	8.8.8.8	80
8.8.8.8	80	10.12.13.14	300
10.12.13.14	451	128.132.53.14	80
128.132.53.14	80	10.12.13.14	451

Deuxième jeu de données			
IP source	Port source	IP destination	Port destination
10.12.13.14	342	8.8.8.8	80
8.8.8.8	80	10.12.13.14	342
10.12.13.14	532	128.132.53.14	80
128.132.53.14	80	10.12.13.14	532
10.12.13.14	576	128.132.53.12	80
128.132.53.11	80	10.12.13.14	576

De ces données, l'algorithme enlève les ports puisque ceux-ci s'interchangent et lorsqu'il s'agit de celui de la machine virtuelle, il est généré aléatoirement par le système d'exploitation.

Les deux tableaux que notre algorithme tri sont représentés au tableau 4.3

Tableau 4.3 Les tableaux que l'algorithme compare

Premier tableau	
IP source	IP destination
10.12.13.14	8.8.8.8
8.8.8.8	10.12.13.14
10.12.13.14	128.132.53.14
128.132.53.14	10.12.13.14

Deuxième tableau	
IP source	IP destination
10.12.13.14	8.8.8.8
8.8.8.8	10.12.13.14
10.12.13.14	128.132.53.14
128.132.53.14	10.12.13.14
10.12.13.14	128.132.53.12
128.132.53.11	10.12.13.14

L'algorithme compare le tableau 2 au tableau 1 et seulement les deux dernières lignes du tableau ne se retrouvent pas dans le premier tableau (*10.12.13.14* et *128.132.52.12* ; *128.132.52.12* et *10.12.13.14*). Dans ce cas, la liste de différences contiendra 2 éléments. Le calcul de similarité divise la taille de cette liste par la taille du plus grand tableau ($\frac{2}{6}$). Notre exemple retourne que la session 1 et session 2 ont une similarité de *66%*(1- $\frac{2}{6}$).

Test sur la durée de vie des pages

La deuxième expérience que nous réalisons cherche à établir la durée de vie moyenne des pages qui sont utilisées pour des infections. Pour ce faire, nous avons modifié la fonction du planificateur pour nous ajuster à notre test. Lors de chaque exploration, une adresse IP différente est choisie. Comme nous n'avons que 7 adresses, si elles ont toutes été utilisées, nous prenons une adresse aléatoirement. La grande différence avec le test de la géolocalisation est l'utilisation du planificateur des prochaines visites.

Données : url

Résultat : Prochain temps d'exploration d'une URL

si *estEnVie(url)* **alors**
  explore(url)
  prochainTempsExploration = planifierProchaineExploration(url, vivante=Vrai)
fin
sinon si *explorationHorsLigne++* < *maxExplorationHorsLigne* **alors**
  prochainTempsExploration = planifierProchaineExploration(url, vivante=Faux)
fin
fileUrl.inserer(url, prochainTempsExploration)

Algorithme 2: Algorithme de calcul du temps de la prochaine exploration

Après une exploration, l'URL visitée est remise dans la file avec une heure de visite 10 minutes plus tard que le temps présent. Le délai de 10 minutes est modifiable, mais nous

avons choisi d'utiliser ce sursis puisque selon [36], les pages web utilisées pour infecter des gens auraient un temps moyen d'utilisation de 2.5 heures. Nous serons donc capables de visiter la page à plusieurs reprise avant que celle-ci ne soit fermée. Nous planifions aussi des visites même si la page est considérée comme morte. Cependant, le délai de visite d'une page hors ligne est de 24 heures puis celle-ci ne sera visitée qu'un maximum de 3 fois si elle est toujours morte. Le but ici étant de déterminer si des pages sont mises hors ligne pour un instant puis remise en ligne par la suite. Ceci est possible si la page qui nous est fournie est victime d'une campagne de publicité malicieuse, par exemple. Dans ce cas, la page peut être mise hors ligne le temps d'enlever la publicité fautive, puis remise en ligne par la suite. Notre système devrait détecter alors que celle-ci n'infecte plus de clients. Il est important de noter que les critères d'infection que nous déterminons dans la section 4.2 sont cruciaux puisqu'il est possible qu'une page reste en ligne longtemps si elle n'est plus utilisée pour infecter des victimes. Nous devons donc déterminer les pages qui ont déjà contaminé des usagers puis vérifier leur durée de vie.

Test sur les règles de liste noire

Le dernier test que nous exécutons nous permet de voir si les gens déployant des virus utilisent un système de liste noire pour éviter d'infecter des gens à plusieurs reprises de façon à réduire le risque de détection. En effet, l'hypothèse que nous posons est que, dans le cas spécifique des kits d'exploitation, un système central gère les victimes et garde une liste de toutes les adresses IP qui ont été infectées précédemment. Dans cette optique, nous avons utilisé une configuration du planificateur très semblable à celle du test de durée de vie des pages. La seule différence est que l'adresse utilisée à chaque exploration reste la même. Encore une fois, les résultats de ce test sont intimement liés avec les critères d'infections que nous définissons à la section 4.2. En effet, si nous considérons que les serveurs centraux des kits d'exploitation font un tri sur les usagers qu'ils décident (ou non) d'infecter, il est probable que les 7 adresses que nous disposions au début des expérimentations soient dans la liste noire. C'est pourquoi nous devons simplement analyser les pages qui, pour la première visite durant ce test, infectent notre client.

4.2 Détermination des critères d'infections

Afin de déterminer si une exploration génère une infection dans une machine virtuelle, nous avons utilisé les métriques suivantes :

1. Binaire(s) téléchargé(s) ou exécuté(s)
2. Fichier *PDF* ou *Flash* téléchargé

3. Machine virtuelle *Java* lancée
4. Aucun fichier du moniteur de processus généré

Ces métriques, bien qu'elles puissent être liées, présentent un certain taux d'erreur qui est difficile de calculer. En effet, comme il sera expliqué dans la dernière sous-section, certains problèmes et techniques nous portent à croire que ces métriques génèrent de faux positifs et de faux négatifs.

4.2.1 Binaire(s) téléchargé(s) ou exécuté(s)

Malgré l'évolution des techniques utilisées par les pirates informatiques pour infecter des gens, il reste toujours que la méthode la plus classique d'infection se fait à l'aide d'un fichier binaire exécutable. En effet, avec des techniques de téléchargement forcés, certaines des pages web visitées par notre explorateur chargent directement un binaire puis l'exécutent pour finaliser une infection. Notre système permet de détecter un téléchargement de binaire en utilisant deux méthodes :

- Analyse de l'entête des requêtes HTTP
- Analyse des fichiers téléchargés dans la trace réseau

Dans le cas de l'analyse des requêtes HTTP, les réponses venant de serveur peuvent contenir un champ *Content-Type* qui définit le type MIME de celle-ci. Ces types, définis par le standard RFC2046[30], permettent d'identifier si un fichier exécutable est envoyé en réponse. Typiquement, si la valeur du champ est *application/octet-stream*, ceci indique une réponse contenant des données binaires arbitraires. Pour des fichiers exécutables, le serveur met normalement ce champ. Il est important de noter que le champ n'est non seulement pas obligatoire, mais il est possible pour un pirate de modifier ce champ pour indiquer autre chose complètement. C'est pourquoi notre système analyse aussi tous les fichiers chargés avec la trace réseau. Chacune des explorations de pages effectuées par nos machines virtuelles sauvegarde la trace réseau sous forme de fichier PCAP. En traitant cette trace, il est possible d'extraire chaque fichier téléchargé durant l'exploration et, grâce à notre module de traitement de fichiers, nous pouvons déterminer si des fichiers de type PE sont téléchargés. Ces deux méthodes permettent d'identifier si un fichier binaire est téléchargé lors d'une exploration et, puisque notre système ne simule aucune interaction humaine, nous pouvons considérer que la page a un comportement malveillant.

4.2.2 Fichier PDF ou Flash téléchargé

Tel qu'il a été expliqué à 4.1.1 la source d'URL potentiellement malicieuse à explorer fournie par ESET contient plusieurs types d'infections parmi lesquelles se trouvent des URL qui

infectent les visiteurs grâce à des exploits PDF. En effet, l'exploitation de documents PDF est la méthode la plus répandue d'attaque ciblée par document infecté avec plus de 75% du total des attaques en 2011 selon le rapport mensuel de Symantec[44, 109]. Par ailleurs, les attaques par documents PDF sont des moyens fréquemment utilisés par les kits d'exploitation pour infecter leurs victimes[48, 26]. Sachant que notre source d'URL pointe sur beaucoup d'URL gérées par des kits d'exploitation, si un fichier PDF est téléchargé sans aucune interaction de l'utilisateur, nous pouvons supposer que celui-ci a été infecté ou du moins qu'un comportement malveillant s'est produit. Pour déterminer si un fichier PDF est téléchargé, nous procédons avec les mêmes techniques que pour les fichiers binaires exécutables. Pour détecter des fichiers PDF dans la trace réseau, nous utilisons toujours le champ *Content-Type*, cette fois, c'est plutôt une valeur parmi celles-ci qui nous indique si le fichier est de type PDF :

- *application/pdf*
- *application/x-pdf*
- *application/x-bzpdf*
- *application/x-gzpdf*

Dans cette métrique, nous avons aussi inclus les fichiers de type *Flash* pour élargir notre plage de résultats. En effet, selon plusieurs rapports[16, 81], les failles du populaire objet d'aide au fureteur de Abobe restent un vecteur d'attaque très utilisé par les développeurs de kits d'exploitation. Typiquement, il n'est pas nécessaire pour un usager d'interagir avec une page web pour télécharger un fichier Flash puisque ceux-ci sont utilisés pour lui afficher du contenu. Par contre, étant donné la nature des URL qui nous sont fournies, nous supposons que lorsqu'une page distribue du contenu Flash à un usager, c'est dans le but de l'infecter. Pour déterminer qu'un fichier Flash est téléchargé, nous procédons de la même manière que pour les fichiers de type PDF, cette fois en utilisant le champ *Content-Type* :

- *application/x-shockwave-flash*

4.2.3 Machine virtuelle *Java* lancée

Un autre vecteur d'attaque très utilisé depuis quelques années est sans aucun doute l'exploitation de failles de sécurité dans la machine virtuelle de Java[94]. En effet, plusieurs compagnies d'antivirus rapportent de nouvelles failles de sécurité dans Java et leur intégration dans différents kits d'exploitation[40, 42, 90, 48], [39] rapportait déjà en 2007 une très grande utilisation de Java comme vecteur d'attaque avec des statistiques de la Microsoft Malware Protection Center[64]. Sachant ceci, la détection de l'utilisation de Java lors d'une exploration de page nous amène à croire que l'usager a possiblement subi une infection. Pour détecter si la machine virtuelle de Java a été lancée, nous parcourons la liste des processus exécutés générée par *Process Monitor* pour identifier le processus *Java.exe* qui représente la

machine virtuelle de Java. Dans ce cas, nous savons que la machine virtuelle est lancée et nous considérons que l'usager est infecté.

4.2.4 Aucun fichier du moniteur de processus généré

Comme il a été vu dans la section 3, notre système d'exploration génère un fichier contenant les informations du moniteur de processus. Ce fichier nous permet d'analyser les différentes actions que la machine virtuelle a effectuées lors d'une exploration. Par contre, dans certains cas, le fichier du moniteur de processus n'est pas transféré vers la machine hôte. En analysant spécifiquement ce comportement, nous avons réalisé que dans le cas où la machine se fait infecter par un rançongiciel[106, 11], la machine se retrouve dans un état présentant une page qui demande une rançon pour débloquer celle-ci. Dans cet état, notre système ne peut transférer le fichier du moniteur de processus et nous devons forcer la fermeture de la machine. Ce type d'attaque connait un gain important en popularité comme le rapportent certaines compagnies d'antivirus[34, 58]. Comme les informations que nous possédons sur l'exploration sont plus limitées dans le cas où le fichier du moniteur de processus n'est pas présent, une analyse de la trace peut nous aider à déterminer s'il y a eu infection ou pas. Cette métrique, quoique moins exacte, permet d'établir si une page distribue des fichiers malveillants. Les limitations seront discutées dans la prochaine sous-section.

4.2.5 Limitations des critères

Les critères définis à la section 4.2 nous permettent d'évaluer, avec une certaine incertitude, si l'exploration d'une page génère une infection de la machine virtuelle. Ces critères ne sont pas complètement indépendants et nous les utilisons de façon complémentaire pour augmenter la précision des résultats. En analysant une session, il est possible que plusieurs de ces métriques soient vraies. Dans ce cas, cela indique simplement que le pirate utilise plusieurs vecteurs d'attaque. Ceci est souvent vrai dans le cas des fichiers PDF qui souvent vont exécuter du code par la machine virtuelle de Java. Cette sous-section discute de différentes limites qu'imposent les critères ainsi que leurs impacts sur nos résultats finaux.

Chiffrement des communications

Selon nos résultats préliminaires, nous avons remarqué que quelques-uns des sites visités cryptaient les communications qui se font avec l'usager. L'utilisation du chiffrement dans les communications permet aux sites distributeurs de virus d'éviter d'être détectés par l'analyse de traces réseau. Avec notre système, il est donc impossible d'analyser le trafic réseau pour déterminer si un fichier exécutable, PDF ou Flash est envoyé à l'usager. Ceci limite

donc notre détection d'infection. Par contre, comme nous utilisons le moniteur de processus conjointement avec la trace réseau, il serait tout de même possible pour notre système de détecter si un fichier exécutable, PDF ou Flash a été écrit sur le disque dur.

Téléchargement de fichier (binaire, Flash ou PDF)

Un autre critère utilisé pour détecter si un certain site distribue des logiciels malveillants est de vérifier les fichiers téléchargés pour détecter si un fichier PDF, Flash ou un exécutable n'a pas été envoyé à l'usager. Cette technique permet de savoir si un fichier est téléchargé, mais n'assure pas que celui-ci est exécuté, et donc qu'il infecte le client. Par contre, nous pouvons tout de même assurer que le site produit un comportement malicieux puisque notre système ne génère aucune interaction client. Ceci implique qu'un usager qui atterrit sur cette page téléchargera automatiquement un fichier, ce qui est un comportement malicieux, sauf dans le cas du fichier Flash.

Aucun fichier du moniteur de processus

Dans le cas où le site visité distribue des rançongiciels, nous avons noté que la machine virtuelle ne s'arrêtait pas correctement, ce qui empêche nos scripts de déplacer le fichier du moniteur de processus. Pour diminuer l'erreur, ce critère est couplé avec la détection de fichier exécutable, PDF ou avec l'exécution de la machine Java. Ceci doit être fait pour éviter les faux positifs puisque nos scripts d'exploration de pages de notre système ne sont pas sans faille et peuvent générer des exécutions sans fichier de moniteur de processus. Si toutefois nous détectons des fichiers malveillants téléchargés grâce à la trace, l'erreur diminue grandement.

Erreur dans la réponse du serveur

Finalement, une dernière limitation de l'utilisation des critères est reliée à la réponse que nous fournissent les serveurs contactés. En effet, nos critères de détection d'infection (ou de non-infection) se basent beaucoup sur les réponses fournies par le serveur. Dans le cas des fichiers téléchargés, nous vérifions l'entête de la réponse pour déterminer si un fichier de type malicieux est distribué à l'utilisateur. Sinon, pour détecter qu'un usager n'est pas infecté, nous regardons la réponse du serveur qui, dans le cas où une réponse avec un statut d'erreur est émise, nous indique que le serveur ne distribue pas de virus. Ceci est vrai dans le cas où les réponses du serveur ne sont pas altérées. En effet, il est assez facile pour un pirate de modifier les réponses du serveur pour laisser croire à l'usager que la page n'existe plus, par exemple. Par contre, dans le cas où le serveur envoie une erreur et qu'aucun comportement malicieux

n'est détecté, nous pouvons prétendre que le site n'est pas malveillant. Nous pouvons donc générer de faux positifs avec ce critère.

CHAPITRE 5

Expérimentation et résultats

Ce chapitre discute de différentes expérimentations que nous avons réalisées avec les résultats obtenus. Nous présenterons chacune des trois expériences que nous avons réalisées avec les résultats obtenus.

5.1 Test de similarité par rapport à la localisation de l'usager

Lors de cette expérience, notre système a effectué près de 5000 explorations sur plus de 1700 URL distinctes sur une période d'environ une semaine. Chaque URL peut être visitée à plusieurs reprises avec seulement l'adresse IP qui diffère, dans le but de déterminer si les pirates informatiques se servent de la localisation de leur victime pour générer des infections différentes. Une des raisons qui motivent cette expérience est la popularité grandissante des rançongiciels qui sont reconnus pour générer une image de rançon spécifique à la localisation de l'usager[85]. Par exemple, le kit d'exploitation Reveton[29] bloque l'ordinateur de l'usager et lui affiche une page lui indiquant qu'il doit payer pour le débloquer. La page est personnalisée par des images de forces policières spécifiques à la région de la victime[3]. Tout d'abord, une première analyse des résultats grâce à nos métriques d'infections définies à la section 4.2 nous permet d'évaluer le taux d'infection des pages visitées. Le tableau 5.1 montre les taux d'infections de la première expérience.

Tableau 5.1 Pourcentage du statut des sessions pour la première expérience

Status	% des sessions
Infecté	8.72%
Non infecté	68.48%
Invalide	22.8%

Les sessions sont déclarées comme invalides lorsque la machine s'arrête abruptement et ne produit aucun fichier de trace ou lorsqu'aucune requête vers l'URL n'est faite. De ces infections, nous pouvons déterminer le vecteur d'infection utilisé pour connaitre quel est le type de virus typiquement distribué. La figure 5.1 énumère les différentes infections selon nos métriques. Le total n'est pas nécessairement de 100% puisque certaines infections répondent à plus d'une de nos métriques et possèdent donc plus d'un vecteur.

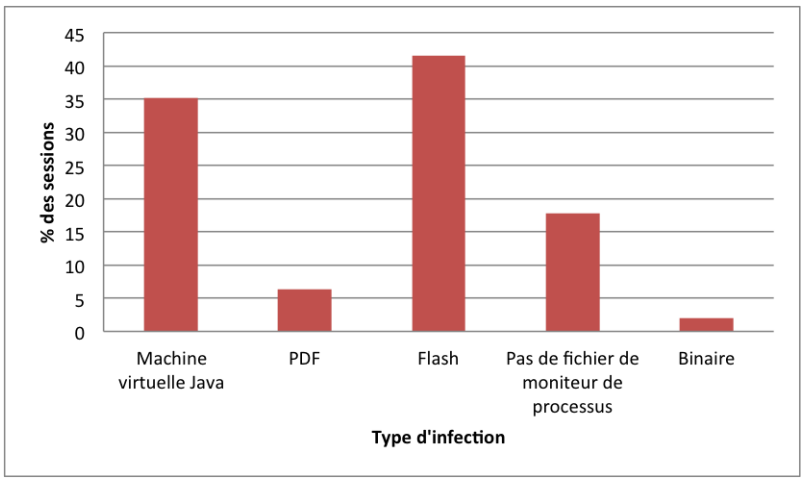


Figure 5.1 Pourcentage des vecteurs d'infection pour la première expérience.

La figure montre bien que le vecteur d'infection le plus utilisé est Flash. Il est important de noter que tout d'abord, toutes les sessions que nous déclarons comme infectées par le vecteur de Flash peuvent posséder un taux de faux positif assez élevé. Malgré le déclin d'utilisation de Flash, il y a toujours près de 17% des sites web qui utilisent présentement Flash[78]. Il est donc probable que parmi les infections rapportées par notre outil, il y ait de faux positifs. Notre métrique pour déterminer si le vecteur d'infection est Java est, lui aussi, tout autant discutable. Par contre, l'utilisation de Java est beaucoup plus faible que Flash, avec 0.1% d'utilisation totale sur le web[79]. Compte tenu de la nature des pages qui nous sont fournies, l'erreur est donc plus faible. Les infections par Java représentent tout de même 35% du total des contaminations. Finalement, le cas où aucun fichier du moniteur de processus n'est créé reste une métrique discutable ; nous en parlerons en détail dans la section 6.1.

Avant de rentrer dans l'analyse détaillée de nos résultats, il est important de déterminer le pourcentage d'infections pour une même URL. C'est à dire, pour une même URL, quel est le pourcentage de sessions qui génèrent des infections. Comme il a été mentionné dans la sous-section 4.1.2, la plage d'adresse IP qui nous est fournie est distribuée sur trois continents sur la planète. Pour ce test, chacune des URL est donc visitée trois fois à partir de chacune des régions. Notre hypothèse est que les sites piratés se comportent différemment si une personne provenant d'Asie s'y connecte versus une personne se connectant d'Europe, par exemple. Il est donc intéressant de calculer combien d'infections sont générées en moyenne pour une même URL. La table 5.2 donne le pourcentage de sessions qui génèrent 1,2 ou 3 infections pour une même URL.

Avec cette table, il semble que les pirates ne font aucune distinction sur l'endroit d'où

Tableau 5.2 Pourcentage des sessions de la même URL qui génèrent une, deux ou trois infections

Nombre d'infections	% des sessions
Une	20%
Deux	11.72%
Trois	68.28%

provient leur victime. En effet, presque 70% des URL qui génèrent des infections le feront sur les trois sessions qui proviennent toutes d'un contient différent. À partir de ceci, il est tout de même intéressant de regarder si une certaine distinction par région est effectuée. Le tableau 5.3 montre les pourcentages d'infections générées par région.

Tableau 5.3 Pourcentage des infections par région

Région	% des sessions
Fremont, Californie	12.26%
Boardman, Oregon	11.42%
Newark, New Jersey	11.70%
Londres, Angleterre	33.15%
Tokyo, Japon	31.47%

Le résultat par région coïncide avec le fait que presque le trois quarts des sessions infectées le feront sans faire de discrimination selon la localisation de leurs victimes. Ceci étant dit, il reste intéressant d'analyser plus précisément les sessions entre elles. Bien que le taux d'infection ne soit pas très élevé (moins de 10%), les sessions qui ont contaminé les usagers possèdent plusieurs informations pertinentes. En effet, en utilisant l'algorithme de similitude entre les sessions décrit à la sous-section 4.1.2, il nous est possible de déterminer les différences entre 2 sessions qui visitent la même URL. Les différences sont séparées en plusieurs sous-groupes basés sur les propriétés provenant de notre objet session tel qu'il a été décrit au chapitre 3. Pour le calcul de la similarité, nous avons pris chaque URL qui génère au moins 1 infection, puis nous avons regardé la similarité entre les autres sessions pour la même URL. Ceci nous permet de tirer des conclusions intéressantes sur nos résultats. Par exemple, la figure 5.2 montre les différentes valeurs que nous avons calculées pour les connexions TCP tel que discuté au paragraphe *Calcul de similarité* de la sous-section 4.1.2.

Les groupes intitulés *infecté* et *non infecté* représentent respectivement les comparaisons de toutes les sessions infectées ensemble et les sessions non infectées ensemble. Le groupe total inclut les deux autres groupes ainsi que toutes les autres comparaisons (sessions infectées et non infectées comparées). Il est donc normal que le groupe total soit plus dispersé étant

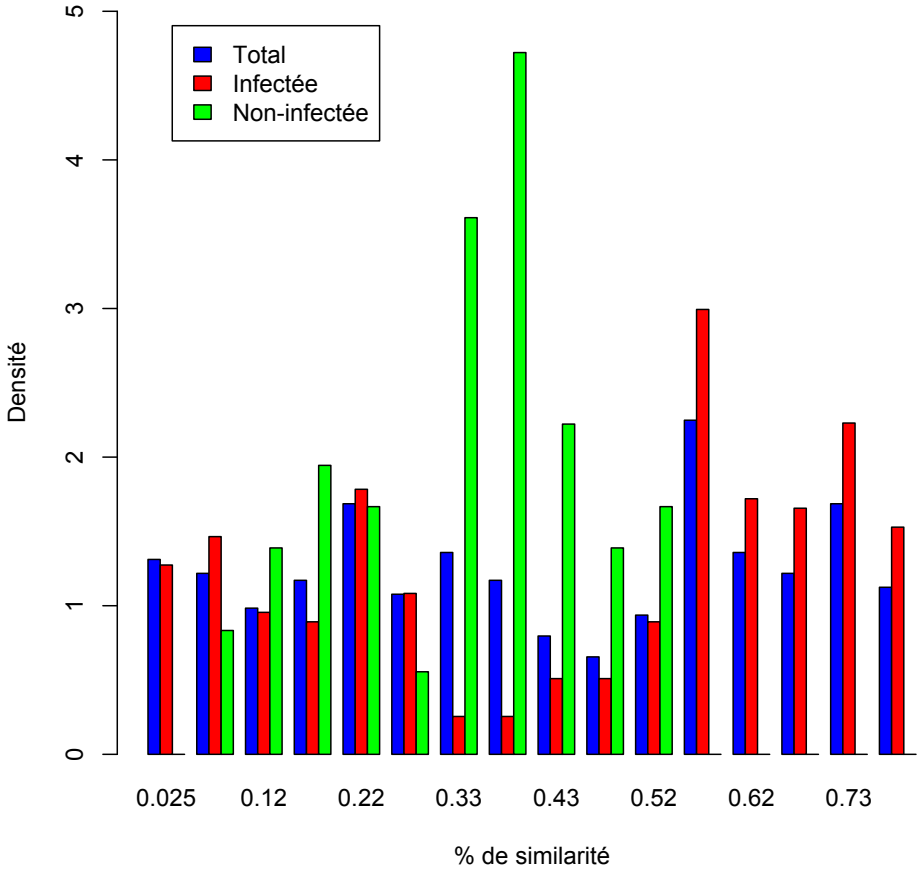


Figure 5.2 Pourcentages normalisés de similitude des connexions TCP par groupe

donné qu'une session infectée comparée à une session non infectée peut contenir beaucoup de différences. Dans le cas des sessions non infectées, nous remarquons qu'elles sont toutes similaires à 55% ou moins avec une grande majorité dans les alentours de 35% de similitude. Ce phénomène est explicable par l'utilisation de balance de charge des serveurs légitime. En effet, comme toutes les sessions proviennent de régions différentes (Europe, Amérique, Asie), les réponses des serveurs DNS ne seront pas les mêmes et donc, les connexions TCP non plus. Si nous comparons aux résultats des sessions infectées, nous remarquons que les sessions ont une grande (55% et plus) ou une faible (25% et moins) similitude. Si la similarité est grande, nous pouvons en déduire que les connexions qui se font au serveur malveillant se font toutes à la même adresse IP, ce qui augmente la similitude du trafic TCP. Toutefois, si la ressemblance du trafic est plus faible, ceci est explicable par le fait que les pirates utilisent des mécanismes de flux rapides de DNS simple ou double pour éviter de se faire détecter ou

qu'ils n'utilisent pas de serveur centralisé de commandes. En effet, ces deux techniques font en sorte que le trafic TCP sera très changeant entre deux sessions. Par contre, les informations recueillies dans le trafic TCP ne démontrent donc pas nécessairement que le contenu distribué est modifié par rapport à la localisation de l'usager.

5.1.1 Analyse spécifique des différents types de similitudes

Malgré que le graphique de similarité des connexions TCP semble indiquer qu'une certaine proportion des sites malveillants (ceux avec un faible taux de similitude) se comportent différemment en fonction de la localisation de l'usager, il est intéressant d'analyser plus précisément les différentes grappes que nous avons. Les prochaines sous-sections se penchent donc sur trois analyses spécifiques de ces groupes que nous décrivons comme :

- Sessions infectées avec un faible taux de similitude TCP
- Sessions infectées avec un fort taux de similitude TCP
- Sessions non infectées avec un taux moyen de similitude TCP

Sessions infectées avec un faible taux de similitude TCP

Selon notre analyse, lorsque le taux de similarité TCP est faible entre deux sessions qui ont visité la même URL, il y a un lien avec l'utilisation de mécanismes tels les flux rapides DNS ou la balance de charge. Pour appuyer nos dires, nous avons pris 4 groupes de sessions qui possèdent un taux de similitude TCP faible (moins de 15%), puis nous les avons analysés. Pour tous les groupes de sessions analysés, nous remarquons que les similarités des requêtes et réponses DNS ne sont pas nécessairement faibles. En effet, des groupes analysés, la table 5.4 montre les différences de similarités pour les requêtes et réponses DNS.

Tableau 5.4 Similarité des réponses et requêtes DNS pour un groupe de trois sessions infectées par la même URL

	Moyenne des similarités	
Groupe	Requêtes DNS	Réponses DNS
1	33.82%	22.98%
2	76.62%	38.96%
3	82.59%	74.74%
4	48.80%	31.28%

En vérifiant les différents groupes, nous arrivons à différents comportements. Tout d'abord, dans le cas où les réponses DNS sont très peu similaires, ceci est le résultat de connexions à des sites légitimes qui retournent des adresses IP de domaines différents en fonction de la localisation. C'est-à-dire que la connexion TCP établie se fera sur une adresse IP différente

en fonction de la localisation du client. La balance de charge est un phénomène très commun et explique cette grande différence de réponse DNS. Dans le cas du troisième groupe, bien que les réponses DNS soient très similaires, le trafic TCP est à 0% tout de même pour toutes les trois sessions. En effet, la réponse DNS retourne une multitude d'adresse IP au client et selon certaines règles du système d'exploitation, celui-ci en choisit une. Notre système d'exploration n'utilise qu'une seule machine qui utilise Windows XP comme système d'exploitation. Il existe deux différentes méthodes pour choisir l'adresse IP lorsqu'une liste est retournée du serveur DNS. La première choisit simplement la première adresse alors que la deuxième utilise un algorithme de sélection. La deuxième méthode est utilisée lorsque la version 6 du protocole IP (IPV6) est installée sur la machine, ce qui est notre cas. L'algorithme boucle sur la liste des adresses IP et choisit celle qui correspond le plus à l'adresse du client en comparant les bits d'adresse. L'article du blogue technique de Microsoft[76] explique l'algorithme plus en détail. Pour les 4 sous-groupes, nous avons déterminé que le groupe *2* et le *4* sont tous les deux des résultats de campagne de distribution de virus. C'est-à-dire que les deux sont des sites légitimes qui auraient distribué des virus à leur insu, probablement par des publicités. Il est tout de même intéressant de regarder le contenu qui est envoyé aux usagers pour savoir si celui-ci diffère aussi. En nous penchant sur le contenu HTTP, nous remarquons que les requêtes, réponses et fichiers sont assez semblables dans presque tous les groupes. La table 5.5 résume les taux de similitudes HTTP des différents groupes de sessions.

Tableau 5.5 Similarité des réponses, requêtes et fichiers HTTP pour un groupe de trois sessions infectées par la même URL

	Moyenne des similarités		
Groupe	Requêtes HTTP	Réponses HTTP	Fichiers HTTP
1	24.67%	24.81%	16.77%
2	51.28%	88.20%	46.59%
3	100%	75%	66.67%
4	30.59%	71.55%	34.74%

Malgré qu'il puisse sembler contre-intuitif que, pour un groupe de sessions, le taux de similitude des requêtes HTTP diffère de beaucoup des réponses, ceci s'explique par le fait que les réponses contiennent moins d'informations et seront donc plus souvent similaires entre sessions. Les pourcentages des similitudes des fichiers HTTP sont un meilleur indicatif. Comme nous pouvons le remarquer, les similarités sont plutôt différentes entre les différents groupes. Encore une fois, les groupes *2* et *4* sont plutôt semblables puisqu'ils sont tous les deux des sites légitimes. Le groupe *1* lui diffère beaucoup, surtout parce qu'une des sessions du groupe est redirigée vers un site légitime (microsoft.com). Ceci change donc le

résultat global des similitudes. Il reste qu'en analysant les groupes *1* et *4*, nous remarquons que les deux semblent se connecter au kit d'exploitation *Blackhole*, le premier groupe se faisant infecter par un fichier Java alors que le deuxième utilise une infection PDF. Si nous regardons les trois traces réseau du groupe *4*, nous remarquons que chacune des sessions a bel et bien reçu un fichier PDF, par contre chacun d'eux est différent. Les pirates utilisent donc certaines méthodes pour modifier leur comportement en lien avec l'usager qui visite leurs sites, par contre, ceci ne semble pas directement lié à la localisation de celui-ci. Ceci n'est pas le cas dans le groupe *1*. Nous ne pouvons pas nécessairement déduire que le comportement d'infection diffère en fonction de la localisation de l'usager. Bien qu'il puisse changer, il est aussi possible que le comportement soit modifié par des variantes légitimes dans le cas d'une campagne publicitaire malicieuse, par exemple.

Sessions infectées avec un fort taux de similitude TCP

Si nous regardons les sessions qui génèrent trois infections pour une même URL et qui possèdent un haut taux de similitude TCP, nous pouvons remarquer que ce groupe n'agit pas exactement comme l'autre. En effet, en analysant les requêtes DNS, nous pouvons voir que celles-ci sont normalement très similaires. La table 5.6 résume les similarités DNS de ce groupe.

Tableau 5.6 Similarité des réponses et requêtes DNS pour un groupe de trois sessions infectées par la même URL

	Moyenne des similarités	
Groupe	Requêtes DNS	Réponses DNS
1	84.84%	63.44%
2	86.31%	41.49%
3	77.17%	36.32%
4	92.77%	78.98%

Les deux groupes *2* et *3* ont des réponses DNS différentes, mais ceci est majoritairement attribuable au fait que les deux font des redirections vers *google.com*, ce qui réduit grandement la similarité. En analysant la trace du groupe *3*, nous remarquons que le site qui infecte l'usager le redirige ensuite vers une page Google. Étant donné que l'usager provient toujours d'un endroit différent sur le globe, les réponses DNS sont extrêmement différentes pour résoudre l'adresse de Google. D'ailleurs, les réponses DNS des domaines malicieux sont, elles, exactement identiques. Le même phénomène se produit pour les données HTTP. En effet, pour ce même groupe, les fichiers HTTP qui sont transmis à l'usager sont presqu'à 100% identiques. Le vecteur d'infection de ce groupe est Java et, en analysant les fichiers

HTTP transférés au client, nous remarquons que le fichier Java qui est transmis au client est exactement le même dans les trois sessions. La table 5.7 résume les différentes similarités entre les données HTTP.

Tableau 5.7 Similarité des réponses, requêtes et fichiers HTTP pour un groupe de trois sessions infectées par la même URL

	Moyenne des similarités		
Groupe	Requêtes HTTP	Réponses HTTP	Fichiers HTTP
1	100%	80%	77.78%
2	94.31%	98.09%	95.70%
3	96%	91.64%	86.12%
4	88.24%	88.24%	78.43%

Ceci valide donc que ce groupe ne semble pas utiliser de mécanisme pour cacher ou modifier leur serveur, mais il distribue aussi le même code malveillant à tous ses visiteurs.

Sessions non infectées avec un taux moyen de similitude TCP

Finalement, le dernier groupe analysé nous permet de déterminer si le comportement d'une page qui ne semble pas infecter d'usager est semblable à celle qui l'infecterait. Typiquement, les sessions qui ne génèrent pas d'infection ont une similarité TCP d'environ 40%. En regardant 4 groupes de sessions qui n'ont pas été infectées et qui ont une similitude TCP moyenne, nous pouvons expliquer leur comportement. Tout d'abord, en regardant les moyennes des similitudes de la table 5.8, nous remarquons que les similitudes s'approchent de celles des sessions infectées avec une grande similarité TCP.

Tableau 5.8 Similarité des réponses et requêtes DNS pour un groupe de trois sessions infectées par la même URL

	Moyenne des similarités	
Groupe	Requêtes DNS	Réponses DNS
1	61.52%	38.36%
2	81.47%	61.31%
3	56.03%	22.37%
4	76.56%	48.30%

Le groupe *2* semble donner des taux de similitudes plus forts que ceux des autres groupes. Ceci s'explique par le fait que cette URL ne semble pas légitime alors que les autres auraient plutôt été des campagnes d'infection temporaire.

Suite à cette analyse, il est difficile de répondre à notre question. En effet, nous avons remarqué qu'il existe deux méthodes distinctes utilisées pour infecter des usagers. Soit les pirates distribuent le virus par un site légitime ou ils utilisent un site web fait explicitement pour l'infection. Dans les deux cas, ils peuvent utiliser des techniques de balance de charge, de flux rapide DNS ou même distribuer des fichiers différents (sûrement grâce à l'obscurcissement) pour modifier le comportement, mais quelques sites ne semblent pas le faire. Avec les informations récoltées, il est donc difficile de sortir des grappes distinctes comportementales.

5.2 Test de durée de vie des pages

Le but de ce deuxième test est d'évaluer la durée de vie des pages web qui distribuent potentiellement du contenu malicieux. Pour ce faire, notre système visite les pages qui lui sont fournies aussi longtemps que la requête DNS nous retourne des adresses IP ou que le serveur ne nous retourne pas d'erreur. Dans le cas où la page retourne une erreur ou que le DNS ne fonctionne plus, la page est revisitée plus tard, jusqu'à un certain nombre de visites tel qu'il a été décrit à la sous-section 4.1.2. Durant cette période de test, plus de 5000 sessions ont été effectuées avec des taux d'infections représentés au tableau 5.9.

Tableau 5.9 Pourcentage du statut des sessions pour la deuxième expérience

Status	% des sessions
Infecté	5.73%
Non infecté	69.05%
Invalide	25.22%

Encore une fois, malgré un taux faible (moins de 6%), les sessions infectées nous donnent assez d'informations pour pouvoir répondre à la question de la durée de vie. Tout d'abord, pour analyser le temps moyen de vie des pages qui contiennent des infections, nous calculons combien de temps une page reste en ligne. Le tableau 5.10 montre les temps de vie minimum, maximum et moyen pour le deuxième test.

Tableau 5.10 Temps de vie des pages qui ont été explorées lors du deuxième test

Temps minimum	Temps moyen	Temps maximum
0 h 17 min 14 s	2 jours, 2 h 02 min 40.147 058 s	3 jours, 18 h 48 min 51 s

Malgré que le test s'exécute sur une période d'environ une semaine, en analysant les résultats, nous remarquons que pour la plupart des URL, la dernière des visites retournait une réponse DNS. Cela signifie que les URL ne sont pas remises en file malgré qu'elles sont encore

vivantes, probablement à cause d'un bogue. Par contre, nous pouvons tout de même interpréter ces résultats et en tirer des conclusions intéressantes. Tout d'abord, sur les presque 5000 sessions d'exploration, près de 12 000 interrogations DNS ont été effectuées sur environ 450 URL uniques. De celles-ci, presque 50% (6123) n'étaient pas en vie et n'ont donc pas généré d'exploration. En fait, seulement 31 des URL ont été visitées, ce qui représente moins de 7% du total. En se basant sur le fait que les courriels nous sont acheminés chaque heure, les domaines auraient donc une période de vie de moins d'une heure. Il est important de noter qu'il se peut que les URL qui nous sont transmises de ESET ne nous parviennent pas directement après une infection détectée préalablement par ESET, par contre ce délai n'est pas une information que nous pouvons calculer.

Si nous effectuons une analyse plus précise des différentes URL qui ont été explorées et restaient en ligne, nous pouvons en soutirer des informations intéressantes. Le tableau 5.11 présente différentes informations sur celles-ci.

Tableau 5.11 Informations sur des interrogations DNS sur les URL explorées

Total de résurrection	Total de domaine mort	Total des URL
20	23	31

Sur le total des URL, 23 fois l'interrogation DNS retournait le domaine comme étant mort. Il est aussi intéressant de voir s'il est possible que des pages ne soient plus vivantes pour un instant puis qu'à la prochaine visite, elles le soient de nouveau. En effet, grâce à notre algorithme d'exploration, nous visitons les pages qui ont déjà été vivantes et qui ne retournent aucune réponse DNS. Par la suite, elles seront visitées de nouveau après un délai minimum de 24 heures. Nous appelons ce phénomène, la résurrection d'une page. Pour des URL qui ont été visitées, à 20 reprises l'interrogation DNS retournait un domaine mort, puis à la prochaine visite elle était vivante de nouveau. Il est intéressant de calculer à quel pourcentage les pages peuvent ressusciter pour nous indiquer si elles font partie d'une campagne sur un site légitime ou si les pirates utilisent des méthodes pour garder une même URL, mais en modifier la destination (information du domaine). Pour déterminer si les pages font partie d'une campagne ou si les pirates utilisent différentes techniques pour modifier la gestion des pages, nous nous intéressons aux données concernant le domaine. Notre système permet de chercher les informations d'un domaine grâce aux outils *WhoIs* et le *WhoIs* du groupe *Cymru* tel qu'expliqué à la sous-section 3.6.3. En récoltant ces informations puis en les comparant entre les diverses sessions qui ont visité les mêmes URL, nous pourrons voir si les pages utilisées pour infecter les usagers sont plutôt légitimes ou pas. Normalement, les données *WhoIs* d'une page ne changent pas très souvent puisqu'elles sont reliées à la date de validation

de l'inscription de la page, le nom de la personne qui l'enregistre et d'autres informations qui ne changent que très rarement. En nous basant sur ceci, si les pages nous donnent un faible taux de similitude pour les informations de domaine, nous pouvons prétendre qu'elles sont sur un domaine qui est entièrement malicieux. Dans le cas contraire, le domaine peut avoir été mis hors ligne le temps de nettoyer les serveurs des infections puis remettre le tout en ligne. Ceci devrait tout de même générer des informations de domaine similaire à 100%. Seulement 4 URL sur le total retournent des informations de domaine (WhoIs uniquement) différentes parmi toutes les sessions. En analysant ces URL, nous remarquons qu'après un certain moment, le registraire de nom de domaine modifie les informations du domaine pour indiquer que celle-ci est victime d'une campagne malicieuse.
Les résultats de ce deuxième test nous permettent difficilement de déterminer avec exactitude le temps moyen des pages qui infectent des usagers. La plupart des pages ne sont pas visitées puisqu'elles sont déjà mortes lorsque nous les recevons, soit dans un délai pouvant aller jusqu'à une heure, en ignorant le délai de ESET. Ceci est d'ailleurs corroboré avec [36] qui a déterminé que le temps moyen de vie d'une page était d'environ 2.5 heures. Sinon, des pages qui sont restées en ligne, quelques-unes (17 URL sur 20 résurrections) retournent temporairement un domaine comme mort puis reviennent en ligne. Ce phénomène que nous avons appelé la résurrection des pages nous porte à croire que les pirates utilisent souvent des pages légitimes pour faire une campagne d'attaque.

5.3 Test de mécanisme de liste noire

Finalement, pour le dernier test, nous tentons de déterminer si les pirates déployant des virus informatiques sur le web utilisent des techniques de liste noire pour éviter de se faire détecter. En effet, l'utilisation d'une liste noire est une technique classique qui permet de filtrer le trafic entrant sur un site web pour éliminer les clients potentiellement malveillants. Dans le cas des pirates, l'utilisation d'un tel système leur permet, par exemple, de ne pas infecter deux fois une machine précédemment infectée, ou tout simplement d'éviter d'infecter des machines qu'ils suspectent être la police ou des chercheurs. Pour cette partie du test, la même adresse IP est donc utilisée pour toutes les explorations. Une première expérience a été réalisée à la suite des deux précédentes, mais comme le système était instable, le taux de sessions invalides était extrêmement élevé ce qui a corrompu l'expérience. Nous avons donc corrigé les problèmes puis effectué les tests de nouveau. Le dernier test a été réalisé sur une période plus courte, soit 2 jours, où plus de 1000 explorations ont été effectuées sur environ 150 URL. Tout comme le deuxième test, les URL peuvent être explorées à plusieurs reprises si le domaine est toujours en ligne. Les résultats d'infection sont plutôt inattendus avec un

taux d'infection d'environ 90% tel que présenté à la table 5.12.

Tableau 5.12 Pourcentage du statut des sessions pour la troisième expérience

Status	% des sessions
Infecté	86.38%
Non infecté	8.56%
Invalide	5.06%

En regardant les sessions infectées, nous avons fait une analyse des vecteurs d'infections. Le graphique 5.3 montre le pourcentage d'infection par vecteur.

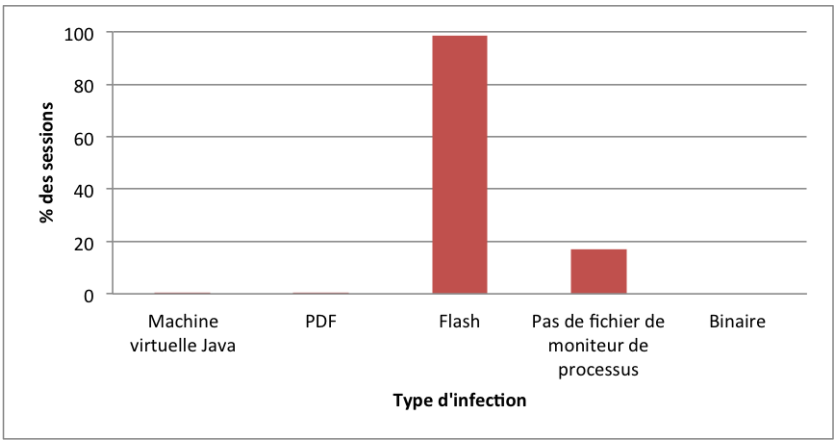


Figure 5.3 Pourcentage des vecteurs d'infection pour la troisième expérience.

Comme nous le remarquons, la majeure partie des infections sont générées par le biais d'une infection Flash. Ceci nous semblait contre-intuitif par rapport aux résultats que nous avions récoltés auparavant. En faisant des inspections manuelles de différentes sessions, nous avons rapidement remarqué que la plupart des nouvelles URL qui nous étaient fournies provenaient de sites aux apparences douteuses, mais qui sont tout de même légitimes. En début octobre 2013, une nouvelle importante a bouleversé le monde du cybercrime alors que le pirate dénommé *Paunch* qui est connu pour être l'auteur derrière le kit d'exploitation *Blackhole* a été arrêté[71, 51, 49]. Bien qu'avec les informations que nous récoltons de notre système il est difficile de déterminer la provenance des infections, nous savions *a priori* que plusieurs d'entre elles étaient liées au kit *Blackhole*. Les derniers résultats obtenus nous poussent à croire que nos URL ainsi que le type de page que nous recevons de notre flux sont modifiés par la disparition ou le ralentissement que connait présentement le monde de la cybercriminalité. Il est possible que ce soit aussi parce nos adresses IP étaient déjà sur les listes noires des pirates et que ceux-ci ne font qu'ignorer nos requêtes. Par contre, les résultats de nos

deux autres premiers tests nous poussent plutôt à valider la première possibilité. D'ailleurs, en vérifiant le temps de vie des pages, la plupart d'entre elles sont en vie sur toute la durée de test.

Tableau 5.13 Temps de vie des pages qui ont été explorées lors du troisième test

Temps minimum	Temps moyen	Temps maximum
0 :00 :00	13 h 45 min 42 s	2 jours, 20 h 52 min 21 s

Le temps minimum de 0 est explicable par l'arrêt de l'exécution des tests avant que plus d'une exploration sur la même URL ne soit effectuée. Sinon, malgré que le temps moyen soit de 13 heures 45 minutes, seulement 1 des URL restera morte, les autres restent toutes en vie sur toute la durée de l'exécution. À la lumière de ces résultats, nous ne pouvons tirer de conclusions quant à l'utilisation ou non d'un système de liste noire par les pirates. Nous pouvons cependant remarquer un changement important dans le déploiement et les méthodes des pirates, ce qui peut être en lien avec les changements que subit présentement l'écosystème de la cybercriminalité.

CHAPITRE 6

Conclusion

En conclusion, l'utilisation d'un système d'exploration de pages, pour tenter de mieux comprendre l'écosystème dans lequel les pirates contribuent maintenant, permet d'augmenter la compréhension de celui-ci et est une solution viable. En effet, les sections 6.1 et 6.2 discuteront des limitations ainsi que des améliorations que nous pourrions apporter au système pour en augmenter l'efficacité et la richesse des données récoltées. Ceci étant dit, avec nos résultats, nous avons su tirer des conclusions intéressantes sur les différents comportements des pirates.

Dans le cadre de ce mémoire, nous avons développé un système en *Python* qui permet d'automatiser l'exploration d'URL potentiellement malicieuse puis d'en tirer le maximum d'information pour ensuite les analyser. Le système a permis d'explorer plus de 13,000 URL uniques sur plus de 10,000 sessions s'étalant sur une période de presqu'un mois. Pour obtenir ces résultats, nous avions établi une méthodologie pour trois tests que nous réalisions dans le but de répondre à nos questions de recherche. Le premier test nous a permis de mieux comprendre le comportement des pirates informatiques par rapport à la localisation de leurs potentielles victimes. Selon nos résultats, il semblerait que les pirates utilisent parfois différentes techniques tels le flux rapide de DNS ou la balance de charge pour décentraliser leur infrastructure, mais que les infections envoyées aux usagers ne diffèrent pas selon la position géographique de celui-ci. Pour le deuxième test, nous tentions d'établir la durée de vie moyenne des pages que les pirates utilisent. La plupart des URL visitées durant ce test sont restées en ligne sur la majeure partie de celui-ci, présentant un faible taux de page morte ou de page ressuscitée. Par contre, les pages retournaient parfois de nouvelles données sur le domaine après un certain moment, presque toujours dans le cas où le registraire de nom de domaine déclarait celui-ci comme potentiellement malicieux. Nous avons aussi remarqué que les pirates utilisent beaucoup des campagnes sur sites légitimes pour maximiser leurs infections. Finalement, le troisième et dernier test cherchait à déterminer si les développeurs de virus utilisaient un système de liste noire pour éviter de se faire détecter. Malgré que ce test ne nous retournait pas de résultats intéressants puisque la plupart des URL que nous avons visitées étaient légitimes et ne semblaient pas infecter leurs utilisateurs, nous avons tout de même pu remarquer une certaine modification comportementale dans les pages qui nous étaient fournies. En effet, le paysage de la cybercriminalité a subi un choc en début

octobre 2013 alors que le développeur du populaire kit d'exploitation *Blackhole* se serait fait arrêter. Comme le flux d'URL envoyé par ESET semblait fortement concentré de pages qui redirigeaient l'usager vers une infection de *Blackhole*, il est fort probable que le changement comportemental observé soit en lien direct avec cet évènement.

Somme toute, nous avons été capables de tirer différentes conclusions, parfois contre-intuitives, sur le comportement des pirates informatiques. Par contre, l'effort à mettre pour continuer de développer un tel système n'en vaut peut-être pas la chandelle considérant la venue d'un projet à code source ouvert d'envergure dénommé *Cuckoo*[19, 72, 86]. Ce projet présente une alternative extrêmement intéressante.

6.1 Limitations de la solution proposée

Dans le chapitre 5, nous avons expliqué les différentes expériences que nous avons réalisées avec les résultats et conclusions que nous en avons tirées. Plusieurs points soulevés limitent ce que nous pouvons conclure, cette section les explique.

6.1.1 Faible taux d'infection

Dans nos trois expériences exécutées, le taux d'infection de nos machines sont toujours assez faible (moins de 10%). Bien que les pages qui nous sont fournies devraient toutes distribuer du contenu malicieux à certains usagers selon ESET, nos expériences semblent dire le contraire. Tout d'abord, étant donné que les pages proviennent d'un logiciel de sécurité qui n'est pas parfait, certaines de ces pages peuvent être de faux positifs. Par contre, le problème majeur est que nos métriques d'évaluation d'infection ne sont pas extrêmement précises. Sur une machine d'un usager normal, il est déjà assez difficile d'évaluer si une visite de page infecte le client, mais la difficulté est beaucoup plus grande étant donné la complexité de notre système combiné à l'utilisation de la virtualisation. Même si nos métriques sont très permissives, il est très possible qu'elle ne soit pas capable d'évaluer certains types d'infections.

6.1.2 Limitations des informations récoltées

Une autre des limitations que nous soulevons est reliée aux informations que nous récoltons durant une exploration. Tout d'abord, le moniteur de processus récolte une multitude d'informations, beaucoup d'entre elles représentent des opérations légitimes. Étant donné l'ampleur des données reçues, il est vraiment difficile de distinguer les opérations malicieuses des opérations légitimes. De plus, une fois la machine virtuelle arrêtée, aucune des données ne

peut être extraite de celle-ci. Donc, même si nous détections une création de fichiers potentiellement malicieux par le moniteur de processus, nous ne pourrions pas extraire ce fichier. Dans le cas de la trace de réseau, elle nous permet de voir tout ce qui se passe lors d'une exploration en terme de réseautage. Le problème est que la plupart des pirates utilisent maintenant des protocoles de chiffrement pour envoyer leurs données. Encore une fois, ceci nous empêche d'extraire les fichiers qui sont la source de l'infection.

6.1.3 Stabilité du système

Durant nos expériences, nous avons remarqué qu'un pourcentage important des sessions d'exploration était invalidé. Une session est invalidée lorsqu'aucune trace réseau n'a pu être générée à cause d'erreurs ou lorsque la machine virtuelle s'arrête abruptement. Malgré que cela ne biaise pas nos résultats, il en reste que l'efficacité de notre système se voit grandement affectée. Un des plus gros problèmes est l'utilisation de Virtual Box par des commandes directement au lieu de passer par leur interface de programmation. Ce choix a été fait puisque l'interface de programmation offerte nous limitait dans ce que nous pouvions faire comme opérations. Par contre, il serait intéressant de voir ce qu'il en est aujourd'hui pour l'utiliser afin d'assurer une plus grande stabilité, et donc augmenter l'efficacité du système.

6.1.4 Complexité de l'écosystème des pirates

En évaluant les résultats d'exploration des URL à travers nos explorations, nous avons rapidement constaté que les pirates informatiques utilisent des systèmes complexes pour distribuer des virus et infecter des millions d'usagers. En effet, étant donné la multitude de failles qui sont exploitables, ou les différentes caractéristiques usagers, les pirates informatiques possèdent une très grande gamme de méthodes pour infecter leurs usagers. Notre système se veut plutôt une preuve de concept pour tenter d'évaluer la faisabilité d'un projet d'envergure pour évaluer de façon la plus complète possible cet écosystème dans lequel plusieurs personnes s'entraident. La prochaine section décrit les améliorations futures que nous pouvons apporter à notre système pour obtenir une plus vaste gamme de résultats.

6.2 Améliorations futures

Le système que nous avons développé pour explorer différentes pages, afin de mieux comprendre l'environnement des pirates, est *a posteriori* petit, étant donnée l'ampleur de la tâche. Dans cette bataille de David contre Goliath, les prochaines sous-sections tentent d'énumérer certaines améliorations que nous pourrions apporter à notre système pour le rendre plus efficace.

6.2.1 Plage d'adresses IP

Puisque les pirates peuvent, et parfois utilisent différentes techniques pour infecter différemment deux usagers de provenances diverses, il nous faudrait une plage d'adresses IP assez grande pour couvrir les endroits majeurs sur le globe. De plus, comme des techniques de liste noire semblent utilisées, il faudrait que cette plage soit dynamique et donc modifiable facilement pour éviter que les pirates ne nous détectent.

6.2.2 Configuration machine

Pour augmenter la diversité de nos résultats, il serait intéressant de faire varier les configurations des machines virtuelles pour en évaluer l'incidence sur les résultats. En effet, il est évident que les pirates analysent la configuration de la machine qui visite leur page pour déterminer quelle faille exploiter sur celle-ci. Une machine peut être modifiée en terme de configurations sous plusieurs formes, allant des modules d'aide au fureteur jusqu'au système d'exploitation, il existe pratiquement une infinité de variations d'une machine. La gestion et la création de celle-ci ne sont pas triviales, mais cela apporterait une plus grande variété aux résultats.

6.2.3 Parallélisme d'exploration

En ce moment, le système que nous avons développé n'utilise pas la puissance du parallélisme pour lui permettre d'envoyer plusieurs explorations simultanément. Pour des raisons de simplifications logicielles, nous avons volontairement décidé de ne pas développer le système de façon parallèle. Avec l'utilisation de grappes d'ordinateur, par exemple, nous pourrions facilement effectuer plusieurs dizaines, voire des centaines d'explorations de façon simultanée, ce qui augmenterait grandement le nombre de sessions infectées que nous récolterions. Ceci représente cependant un défi de taille en terme de génie logiciel que nous ne pouvions réaliser dans les délais.

6.2.4 Augmentation des informations récoltées

Finalement, il aurait été intéressant d'augmenter la plage d'informations récoltées de chacune des machines virtuelles après une exploration. Nous pourrions penser à quelque chose d'aussi simple qu'une prise d'écran avant la fermeture pour déterminer si ce que l'usager voit est semblable. Une copie de la mémoire et des registres pourrait aussi être prise en début et en fin d'exploration pour pouvoir déterminer les différentes opérations que le système a subies. Pour une meilleure analyse *a posteriori*, nous pourrions sauvegarder des images du disque de chaque exploration pour la comparer à l'image initiale avant la visite.

Ceci ne sont que quelques exemples qui permettraient, non seulement d'augmenter l'efficacité de notre système, et aussi la diversité des résultats ainsi que leur exactitude.

RÉFÉRENCES

[1] Ross Anderson, Chris Barton, Rainer Böhme, Richard Clayton, Michel JG van Eeten, Michael Levi, Tyler Moore, and Stefan Savage. Measuring the cost of cybercrime. In *11th Workshop on the Economics of Information Security (June 2012)*, 2012.

[2] Michael Bailey, Jon Oberheide, Jon Andersen, Z Morley Mao, Farnam Jahanian, and Jose Nazario. Automated classification and analysis of internet malware. In *Recent Advances in Intrusion Detection*, pages 178–197. Springer, 2007.

[3] David Balaban. "your pc is blocked" : Background of the police ransomware virus | privacy pc [online]. September 2013. URL : `http://privacy-pc.com/news/your-pc-is-blocked-background-of-the-police-ransomware-virus.html` [cited 2013-10-16].

[4] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11. Citeseer, 2009.

[5] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. A view on current malware behaviors. In *USENIX workshop on large-scale exploits and emergent threats (LEET)*, 2009.

[6] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. Ttanalyze : A tool for analyzing malware. In *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, 2006.

[7] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[8] Yuval Ben-Itzhak. Welcome to the malware-as-a-service business model [online]. 2013. URL : `http://blogs.avg.com/news-threats/malware-as-a-service-business-model/` [cited 2013-11-09].

[9] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7) :422–426, 1970.

[10] Lorenzo Cavallaro, Christopher Kruegel, Giovanni Vigna, Fang Yu, Muath Alkhalaf, Tevfik Bultan, Lili Cao, Lei Yang, Heather Zheng, Christopher C Cipriano, et al. Mining the network behavior of bots. Technical report, Tech. Rep. 2009-12, Department of Computer Science, University of California at Santa Barbara (UCSB), CA, USA, 2009.

[11] Microsoft Malware Protection Center. Microsoft malware protection center - ransomware [online]. 2013. URL : http://www.microsoft.com/security/portal/mmpc/shared/ransomware.aspx [cited 2013-09-19].

[12] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. *Proceedings of the Usenix Security*, 2003.

[13] Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. Mining specifications of malicious behavior. In *Proceedings of the 1st India software engineering conference*, pages 5–14. ACM, 2008.

[14] Mihai Christodorescu, Somesh Jha, Sanjit A Seshia, Dawn Song, and Randal E Bryant. Semantics-aware malware detection. In *Security and Privacy, 2005 IEEE Symposium on*, pages 32–46. IEEE, 2005.

[15] Fred Cohen. Computer viruses - theory and experiments [online]. 1984. URL : http://web.eecs.umich.edu/~aprakash/eecs588/handouts/cohen-viruses.html [cited 2013-10-02].

[16] Contagio. contagio : An overview of exploit packs (update 19.1) april 2013 [online]. 2013. URL : http://contagiodump.blogspot.ca/2010/06/overview-of-exploit-packs-update.html [cited 2013-10-10].

[17] Symantec Corporation. 2012 norton cybercrime report. Technical report, September 2012.

[18] André D. Corrêa. Malware patrol [online]. 2013. URL : https://www.malwarepatrol.net/ [cited 2013-10-09].

[19] Cuckoo. Cuckoo sandbox - open source automated malware analysis [online]. 2013. URL : https://media.blackhat.com/us-13/US-13-Bremer-Mo-Malware-Mo-Problems-Cuckoo-Sandbox-WP.pdf [cited 2013-11-06].

[20] Team Cymru. Malware infections market. Technical report, Team Cymru, 2010.

[21] Team Cymru. Team cymru ip to asn lookup v1.0 [online]. 2013. URL : http://whois.cymru.com/ [cited 2013-10-01].

[22] Inc. Damballa. 3% to 5% of enterprise assets are compromised by bot-driven targeted attack malware – atlanta, march 2 /prnewswire/ – [online]. 2013. URL : http://www.prnewswire.com/news-releases/3-to-5-of-enterprise-assets-are-compromised-by-bot-driven-targeted-attack-malware-
html [cited 2013-08-06].

[23] Manuel Egele, Engin Kirda, and Christopher Kruegel. Mitigating drive-by download attacks : Challenges and open problems. In *iNetSec 2009–Open Research Problems in Network Security*, pages 52–62. Springer, 2009.

[24] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2) :6, 2012.

[25] ESET. Eset | antivirus, internet security software virus protection [online]. 2013. URL : `http://www.eset.com/` [cited 2013-09-13].

[26] Jose Miguel Esparza. Pdf attack : A journey from the exploit kit to the shellcode. In *Black Hat USA*, 2013.

[27] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[28] Security Information Exchange. Isc security [online]. 2013. URL : `https://sie.isc.org/` [cited 2013-08-19].

[29] F-Secure. Trojan :w32/reveton [online]. 2013. URL : `http://www.f-secure.com/v-descs/trojan_w32_reveton.shtml` [cited 2013-10-16].

[30] The Internet Engineering Task Force. Rfc 2046 - multipurpose internet mail extensions (mime) part two : Media types [online]. 2013. URL : `http://tools.ietf.org/html/rfc2046` [cited 2013-09-11].

[31] Python Software Foundation. Python programming language – official website [online]. 2013. URL : `http://www.python.org/` [cited 2013-10-05].

[32] Matt Fredrikson, Somesh Jha, Mihai Christodorescu, Reiner Sailer, and Xifeng Yan. Synthesizing near-optimal malware specifications from suspicious behaviors. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 45–60. IEEE, 2010.

[33] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[34] Gavin O'Gorman Geoff McDonald. Ransomware : A growing menace. Technical report, Symantec Corporation, 2012.

[35] Marius Gheorghescu. An automated virus classification system. In *Virus bulletin conference*, pages 294–300. Citeseer, 2005.

[36] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, et al. Manufacturing compromise : the emergence of exploit-as-a-service. In

Proceedings of the 2012 ACM conference on Computer and communications security, pages 821–832. ACM, 2012.

[37] The SecDev Group. The secdev group [online]. 2013. URL : `https://www.secdev.com/` [cited 2013-09-28].

[38] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer : Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX Security Symposium*, pages 139–154, 2008.

[39] Peter Gutmann. The commercial malware industry. In *DEFCON conference*, 2007.

[40] Bob Hansmann. Exploit kits : Making instant java attacks (part iii) - websense insights [online]. Août 2013. URL : `http://community.websense.com/blogs/websense-insights/archive/2013/08/26/exploit-kits-making-instant-java-attacks-part-iii.aspx` [cited 2013-09-19].

[41] Heritrix. Heritrix - heritrix - ia webteam confluence [online]. 2013. URL : `https://webarchive.jira.com/wiki/display/Heritrix/Heritrix` [cited 2013-08-15].

[42] Shinsuke Honjo. Multiple java exploits hide in a jar (file) | blog central [online]. Avril 2013. URL : `http://blogs.mcafee.com/mcafee-labs/multiple-java-exploits-hide-in-a-jar-file` [cited 2013-09-19].

[43] Trend Micro Incorporated. Blackhole exploit kit : A spam campaign, not a series of individual spam runs an in-depth analysis. Technical report, Trend Micro Incorporated, 2012.

[44] Symantec.cloud MessageLabs Intelligence. February 2011 intelligence report. Technical report, Symantec Corporation, 2011.

[45] Rafiqul Islam, Ronghua Tian, Lynn Batten, and Steve Versteeg. Classification of malware based on string and function feature selection. In *Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second*, pages 9–17. IEEE, 2010.

[46] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 128–138. ACM, 2007.

[47] John P John, Alexander Moshchuk, Steven D Gribble, and Arvind Krishnamurthy. Studying spamming botnets using botlab. In *NSDI*, volume 9, pages 291–306, 2009.

[48] Jason Jones. The state of web exploit kits. Technical report, HP DVLabs, 2011.

[49] John P. Mello Jr. 'paunch' arrest puts blackhole hackers on data diet | hacking | technewsworld [online]. 2013. URL : `http://www.technewsworld.com/story/79184.html` [cited 2013-10-05].

[50] Kafeine. The path to infection - eye glance at the first line of "russian underground" - focused on ransomware | malware don't need coffee [online]. 2012. URL : http://malware.dontneedcoffee.com/2012/12/eyeglanceru.html [cited 2013-11-11].

[51] Kafeine. Paunch's arrest...the end of an era ! | malware don't need coffee [online]. 2013. URL : http://malware.dontneedcoffee.com/2013/10/paunch-arrestationthe-end-of-era.html [cited 2013-10-05].

[52] Joris Kinable and Orestis Kostakis. Malware classification based on call graph clustering. *Journal in computer virology*, 7(4) :233–245, 2011.

[53] Johannes Kinder, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. Detecting malicious code by model checking. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 174–187. Springer, 2005.

[54] Kingsoft. Kingsoft office software : free office software, professional office software [online]. 2013. URL : http://www.kingsoftstore.com/ [cited 2013-08-01].

[55] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *USENIX Security Symposium*, pages 351–366, 2009.

[56] Christian Kreibich, Nicholas Weaver, Chris Kanich, Weidong Cui, and Vern Paxson. Gq : Practical containment for measuring modern malware systems. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 397–412. ACM, 2011.

[57] Christopher Kruegel, William Robertson, and Giovanni Vigna. Detecting kernel-level rootkits through binary analysis. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 91–100. IEEE, 2004.

[58] McAfee Labs. Mcafee threats report : Second quarter 2013. Technical report, McAfee, 2013.

[59] Lavasoft. Ad-aware free antivirus and antispyware by lavasoft | protection from virus, spyware malware | top internet security for windows [online]. 2013. URL : http://www.lavasoft.com/ [cited 2013-08-15].

[60] Malware Domain List. Mdl [online]. 2013. URL : http://www.malwaredomainlist.com/ [cited 2013-08-19].

[61] Niels Provos Panayiotis Mavrommatis and Moheeb Abu Rajab Fabian Monrose. All your iframes point to us. 2008.

[62] McAfee. Mobile malware growth continuing in 2013 | mcafee [online]. 2013. URL : http://www.mcafee.com/ca/security-awareness/articles/mobile-malware-growth-continuing-2013.aspx [cited 2013-10-16].

[63] Microsoft. Internet explorer - microsoft windows [online]. 2013. URL : http://windows.microsoft.com/en-ca/internet-explorer/download-ie [cited 2013-08-15].

[64] Microsoft. Microsoft malware protection center home page [online]. 2013. URL : http://www.microsoft.com/security/portal/mmpc/default.aspx [cited 2013-09-19].

[65] Microsoft. Process monitor [online]. 2013. URL : http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx [cited 2013-08-29].

[66] Microsoft. Service pack center - microsoft windows [online]. 2013. URL : http://windows.microsoft.com/en-ca/windows/service-packs-download#sptabs=win8 [cited 2013-09-25].

[67] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421–430. IEEE, 2007.

[68] Alexander Moshchuk, Tanya Bragin, Steven D Gribble, and Henry M Levy. A crawler-based study of spyware in the web. In *NDSS*, 2006.

[69] Mozilla. Mozilla firefox web browser — free download — mozilla.org [online]. 2013. URL : http://www.mozilla.org/en-US/firefox/new/ [cited 2013-08-15].

[70] Arbor Networks. Arbor networks | atlas dashboard : Global [online]. 2013. URL : http://atlas.arbor.net/ [cited 2013-08-19].

[71] RT News. Blackhole malware kingpin 'paunch' arrested in russia — rt news [online]. 2013. URL : http://rt.com/news/blackhole-paunch-arrest-russia-946/ [cited 2013-10-05].

[72] Claudio "nex" Guarnieri Cuckoo Sandbox Developers. Automated malware analysis - cuckoo sandbox [online]. 2013. URL : http://www.cuckoosandbox.org/ [cited 2013-11-06].

[73] Oracle. Oracle vm virtualbox [online]. 2013. URL : https://www.virtualbox.org/ [cited 2013-08-29].

[74] Pierluigi Paganini. The rise of malware as a service (maas) [online]. Février 2013. URL : http://hplusmagazine.com/2013/02/18/the-rise-of-malware-as-a-service-maas/ [cited 2013-04-16].

[75] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*, pages 391–404, 2010.

[76] Michael Platts. Dns round robin and destination ip address selection - microsoft enterprise networking team - site home - technet blogs [online]. 2013. URL : http://blogs.technet.com/b/networking/archive/2009/04/17/dns-round-robin-and-destination-ip-address-selection.aspx [cited 2013-11-11].

[77] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu, et al. The ghost in the browser analysis of web-based malware. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 4–4, 2007.

[78] Q-Success. Usage statistics of flash for websites, october 2013 [online]. 2013. URL : http://w3techs.com/technologies/details/cp-flash/all/all [cited 2013-09-19].

[79] Q-Success. Usage statistics of java for websites, october 2013 [online]. 2013. URL : http://w3techs.com/technologies/details/cp-javaruntime/all/all [cited 2013-09-19].

[80] Orlando Barrera Randy Abrams, Jayendra Pathak. Browser security comparative analysis socially engineered malware blocking. Technical report, NSS Labs, 2013.

[81] DeepEnd Research. Deepend research : Common exploit kits 2012 poster [online]. 2012. URL : http://www.deependresearch.org/2012/11/common-exploit-kits-2012-poster.html [cited 2013-10-10].

[82] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[83] Christian Rossow, Christian J Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten van Steen, Felix C Freiling, and Norbert Pohlmann. Sandnet : Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 78–88. ACM, 2011.

[84] David Sancho. Police ransomware update. Technical report, Trend Micro Incorporated, 2012.

[85] David Sancho and Feike Hacquebord. The “police trojan” an in-depth analysis. Technical report, Trend Micro Incorporated, 2012.

[86] Cuckoo Sandbox. Malwr - malware analysis by cuckoo sandbox [online]. 2013. URL : https://malwr.com/ [cited 2013-11-06].

[87] M Zubair Shafiq, S Momina Tabish, and Muddassar Farooq. Are evolutionary rule learning algorithms appropriate for malware detection ? In *Proceedings of the 11th*

Annual conference on Genetic and evolutionary computation, pages 1915–1916. ACM, 2009.

[88] Monirul Sharif, Vinod Yegneswaran, Hassen Saidi, Phillip Porras, and Wenke Lee. Eureka : A framework for enabling static malware analysis. In *Computer Security-ESORICS 2008*, pages 481–500. Springer, 2008.

[89] Carsten Sulzberger. Efficient implementation of the levenshtein-algorithm, fault-tolerant search technology, error-tolerant search technologies [online]. 2013. URL : `http://www.levenshtein.net/` [cited 2013-04-16].

[90] Jack Tang. Java native layer exploits going up | security intelligence blog | trend micro [online]. Août 2013. URL : `http://blog.trendmicro.com/trendlabs-security-intelligence/java-native-layer-exploits-going-up/` [cited 2013-09-19].

[91] AVG Technologies. What is blackhole exploit kit ? - virus removal and information from avg threat labs [online]. 2013. URL : `http://www.avgthreatlabs.com/virus-and-malware-information/info/blackhole-exploit-kit/#` [cited 2013-08-19].

[92] Ronghua Tian, Lynn Batten, Rafiqul Islam, and Steven Versteeg. An automated classification system based on the strings of trojan and virus families. In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pages 23–30. IEEE, 2009.

[93] Ronghua Tian, Lynn Margaret Batten, and SC Versteeg. Function length as a tool for malware classification. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 69–76. IEEE, 2008.

[94] Gilad Bracha Alex Buckley Tim Lindholm, Frank Yellin. *The Java Virtual Machine Specification, Java SE 7 Edition (Java Series)*. Addison-Wesley Professional ; 1 edition, 2013.

[95] Emil Lindgjerdet Vaagland. *URL Crawling & classification system*. PhD thesis, Norwegian University of Science and Technology, 2012.

[96] VX Vault. Vx vault [online]. 2013. URL : `http://vxvault.siri-urz.net/ViriList.php` [cited 2013-10-09].

[97] Secure Systems Lab Vienna University of Technology. Anubis : Analyzing unknown binaries [online]. 2013. URL : `http://anubis.iseclab.org/` [cited 2013-08-12].

[98] Gérard Wagener, Alexandre Dulaunoy, et al. Malware behaviour analysis. *Journal in computer virology*, 4(4) :279–287, 2008.

[99] GNU Wget. Free software foundation, inc. [online]. 2010. URL : http://www.gnu.org/software/wget/ [cited 2013-10-08].

[100] Whois. Whois.com - domain names identity for everyone [online]. 2013. URL : http://www.whois.com/ [cited 2013-10-01].

[101] Wikipedia. Autonomous system (internet) - wikipedia, the free encyclopedia [online]. 2013. URL : http://en.wikipedia.org/wiki/Autonomous_System_(Internet) [cited 2013-10-02].

[102] Wikipedia. Browser helper object - wikipedia, the free encyclopedia [online]. 2013. URL : http://en.wikipedia.org/wiki/Browser_Helper_Object [cited 2013-09-26].

[103] Wikipedia. Hellinger distance - wikipedia, the free encyclopedia [online]. 2013. URL : http://en.wikipedia.org/wiki/Hellinger_distance [cited 2013-08-05].

[104] Wikipedia. Malware - wikipedia, the free encyclopedia [online]. 2013. URL : http://en.wikipedia.org/wiki/Malware [cited 2013-04-18].

[105] Wikipedia. Phylogenetic tree - wikipedia, the free encyclopedia [online]. 2013. URL : http://en.wikipedia.org/wiki/Phylogenetic_tree [cited 2013-08-05].

[106] Wikipedia. Ransomware (malware) - wikipedia, the free encyclopedia [online]. 2013. URL : http://en.wikipedia.org/wiki/Ransomware_(malware) [cited 2013-09-19].

[107] Wikipedia. Software as a service - wikipedia, the free encyclopedia [online]. 2013. URL : http://en.wikipedia.org/wiki/Software_as_a_service [cited 2013-04-16].

[108] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *Security & Privacy, IEEE*, 5(2) :32–39, 2007.

[109] Paul Wood. The pdf exploit : Same crime, different face | symantec connect community [online]. avril 2011. URL : http://www.symantec.com/connect/blogs/pdf-exploit-same-crime-different-face [cited 2013-09-13].

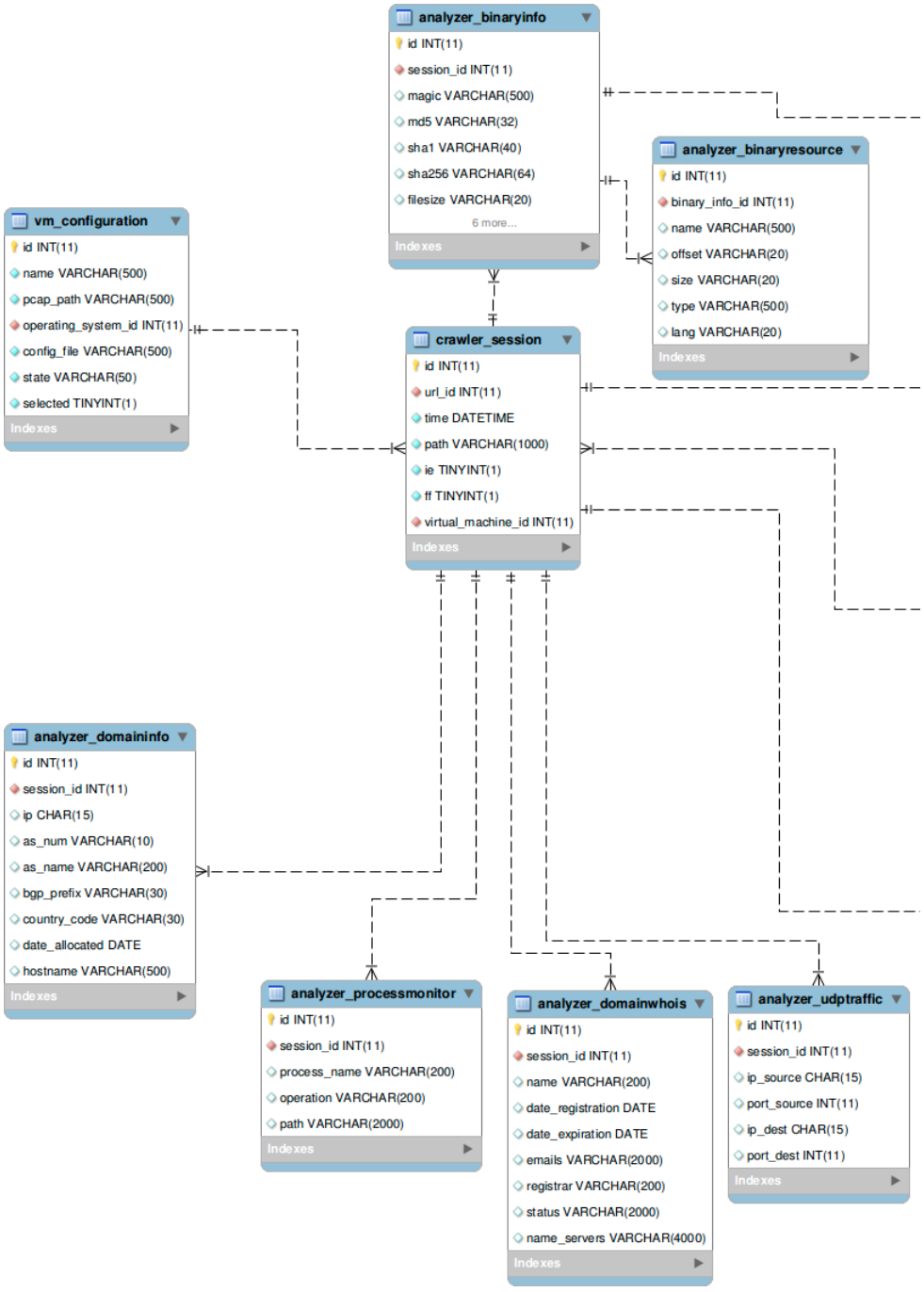
[110] Yanfang Ye, Dingding Wang, Tao Li, and Dongyi Ye. Imds : Intelligent malware detection system. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1043–1047. ACM, 2007.

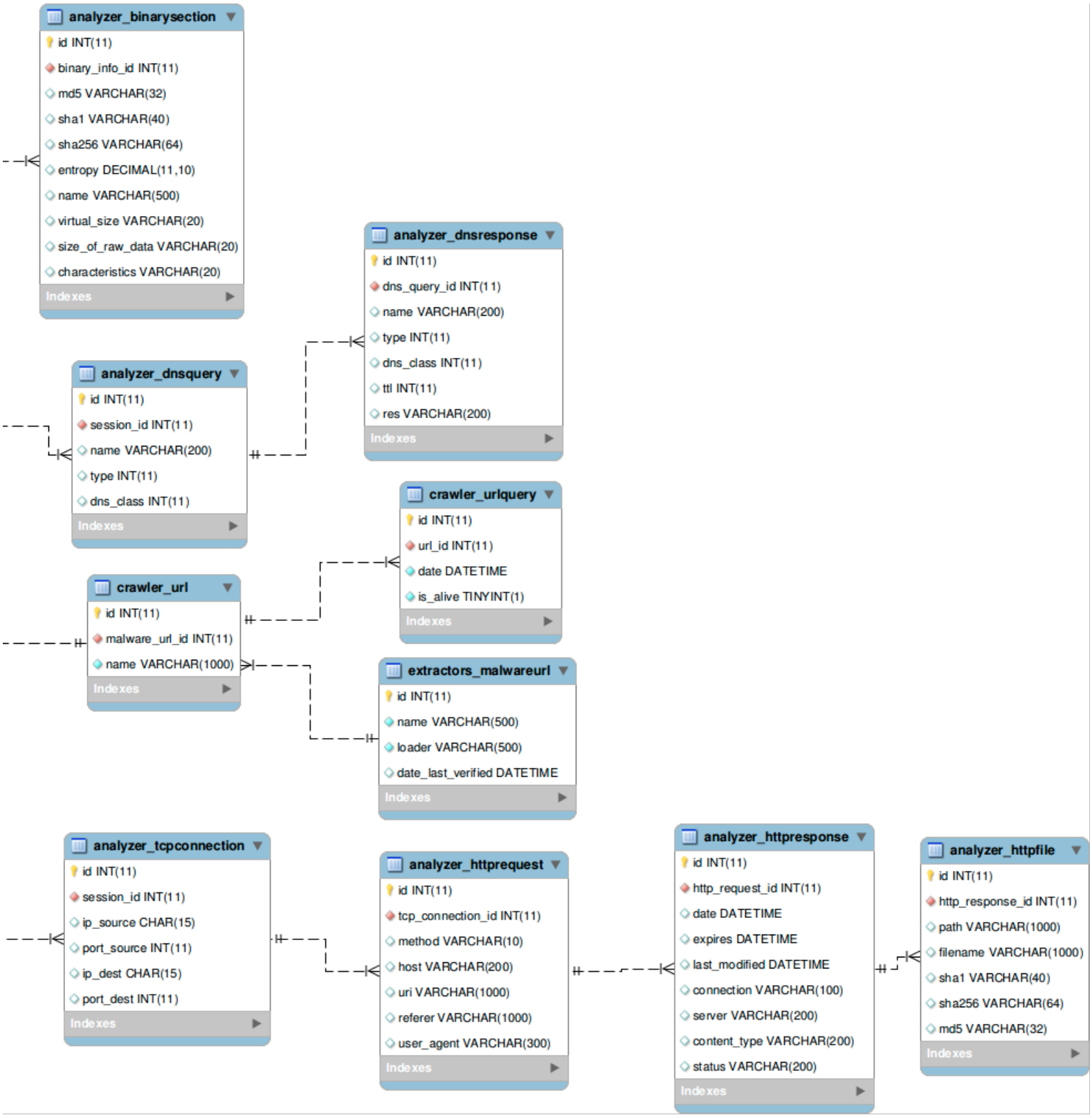
[111] Junjie Zhang, Christian Seifert, Jack W Stokes, and Wenke Lee. Arrow : Generating signatures to detect drive-by downloads. In *Proceedings of the 20th international conference on World wide web*, pages 187–196. ACM, 2011.

[112] Xiangyu Zhang, Rajiv Gupta, and Youtao Zhang. Precise dynamic slicing algorithms. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 319–329. IEEE, 2003.

ANNEXE A

Schéma de la base de données des informations d'exploration




analyzer_binarysection
id INT(11)
binary_info_id INT(11)
md5 VARCHAR(32)
sha1 VARCHAR(40)
sha256 VARCHAR(64)
entropy DECIMAL(11,10)
name VARCHAR(500)
virtual_size VARCHAR(20)
size_of_raw_data VARCHAR(20)
characteristics VARCHAR(20)
Indexes
analyzer_dnsresponse
id INT(11)
dns_query_id INT(11)
name VARCHAR(200)
type INT(11)
dns_class INT(11)
ttl INT(11)
res VARCHAR(200)
Indexes
analyzer_dnsquery
id INT(11)
session_id INT(11)
name VARCHAR(200)
type INT(11)
dns_class INT(11)
Indexes
crawler_urlquery
id INT(11)
url_id INT(11)
date DATETIME
is_alive TINYINT(1)
Indexes
crawler_url
id INT(11)
malware_url_id INT(11)
name VARCHAR(1000)
Indexes
extractors_malwareurl
id INT(11)
name VARCHAR(500)
loader VARCHAR(500)
date_last_verified DATETIME
Indexes
analyzer_tcpconnection
id INT(11)
session_id INT(11)
ip_source CHAR(15)
port_source INT(11)
ip_dest CHAR(15)
port_dest INT(11)
Indexes
analyzer_httprequest
id INT(11)
tcp_connection_id INT(11)
method VARCHAR(10)
host VARCHAR(200)
uri VARCHAR(1000)
referer VARCHAR(1000)
user_agent VARCHAR(300)
Indexes
analyzer_httpresponse
id INT(11)
http_request_id INT(11)
date DATETIME
expires DATETIME
last_modified DATETIME
connection VARCHAR(100)
server VARCHAR(200)
content_type VARCHAR(200)
status VARCHAR(200)
Indexes
analyzer_httpfile
id INT(11)
http_response_id INT(11)
path VARCHAR(1000)
filename VARCHAR(1000)
sha1 VARCHAR(40)
sha256 VARCHAR(64)
md5 VARCHAR(32)
Indexes

ANNEXE B

Schéma de la base de données des machines virtuelles